



VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

ELEKTRONIKOS FAKULTETAS

ELEKTRONINIŲ SISTEMŲ KATEDRA

Vladas TALUNTIS

**TELEMETRINIŲ DUOMENŲ ŠIFRAVIMO ALGORITMO
ĮGYVENDINIMŲ LPLM ĮRENGINIUOSE TYRIMAS**
**INVESTIGATION OF THE FPGA BASED TELEMETRY DATA CODING
ALGORITHM IMPLEMENTATION**

Magistro baigiamasis darbas

Elektronikos inžinerijos studijų kryptis

Elektronikos studijų programa, valstybinis kodas 621H61003

Kompiuterizuotų elektroninių sistemų specializacija

Vilnius, 2017

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
ELEKTRONIKOS FAKULTETAS
ELEKTRONINIŲ SISTEMŲ KATEDRA

TVIRTINU
Katedros vedėjas

(parašas)
prof. dr. Dalius Navakas.
2017 m. _____ mėn.
____ d.

Vladas TALUNTIS

**TELEMETRINIŲ DUOMENŲ ŠIFRAVIMO ALGORITMO
ĮGYVENDINIMŲ LPLM ĮRENGINIUOSE TYRIMAS**
**INVESTIGATION OF THE FPGA BASED TELEMETRY DATA CODING
ALGORITHM IMPLEMENTATION**

Magistro baigiamasis darbas

Elektronikos inžinerijos studijų kryptis
Elektronikos studijų programa, valstybinis kodas 621H61003
Kompiuterizuotų elektroninių sistemų specializacija

Vadovas	dr. Tomyslav Sledevič	_____	_____
	(mokslinis vardas ir laipsnis, vardas, pavardė)	(parašas)	(data)
Konsultantas	dr. Aušra Žemienė	_____	_____
	(mokslinis vardas ir laipsnis, vardas, pavardė)	(parašas)	(data)

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
ELEKTRONIKOS FAKULTETAS
ELEKTRONINIŲ SISTEMŲ KATEDRA

Technologijos mokslų sritis
Elektros ir elektronikos inžinerijos mokslo kryptis
Elektronikos inžinerijos studijų kryptis
Elektronikos studijų programa, valst. kodas 621H61003
Kompiuterizuotų elektroninių sistemų specializacija

TVIRTINU
Katedros vedėjas

prof. dr. D. Navakauskas
2016 m. _____ mėn. ____ d.

MAGISTRO BAIGIAMOJO DARBO
UŽDUOTIS

Studentui Vladui TALUNČIUI, EKSfm-15 gr.

Baigiamojo darbo tema: Telemetrinių duomenų šifravimo algoritmo įgyvendinimų LPLM įrenginiuose tyrimas

Investigation of the FPGA Based Telemetry Data Coding Algorithm Implementation

Patvirtinta 2016 m. _____ mėn. ____ d. dekanų įsakymu Nr. _____.

Baigiamojo darbo užbaigimo terminas 2017 m. _____ mėn. ____ d.

Darbo tikslas – LPLM įrenginyje įgyvendinti ir ištirti telemetrinių duomenų šifravimo algoritmą.

Darbo uždaviniai

1. Apžvelgti duomenų šifravimo algoritmus bei jų įgyvendinimo galimybes LPLM įrenginyje.
2. LPLM įrenginyje įgyvendinti duomenų šifravimo algoritmą.
3. Ištirti duomenų šifravimo algoritmą LPLM įrenginiuose.

Aiškinamojo rašto turinys

1. Įvadas
2. Analitinė literatūros apžvalga
3. Duomenų šifravimo algoritmo kūrimas ir įgyvendinimas LPLM įrenginiuose
4. Duomenų šifravimo algoritmo tyrimas
5. Apibendrinimas. Išvados

Literatūra

Priedai

A priedas. CD su magistro baigiamojo darbo aiškinamuoju raštu

B priedas. Pranešimo 20-oje Lietuvos jaunųjų mokslininkų konferencijoje medžiaga

Baigiamojo darbo rengimo konsultantas:

—

(mokslinis vardas ir laipsnis, vardas, pavardė)

Vadovas

(parašas)

dr. Tomyslav Sledevič

(mokslinis vardas ir laipsnis, vardas, pavardė)

Užduotį gavau

(parašas)

2016 m. _____ mėn. ____ d.

Vilniaus Gedimino technikos universitetas
Elektronikos fakultetas
Elektroninių sistemų katedra

ISBN ISSN
Egz. sk.
Data-.....-.....

Antrosios pakopos studijų **Elektronikos inžinerijos** programos magistro baigiamasis darbas

Pavadinimas **Telemetrinių duomenų šifravimo algoritmo įgyvendinimų LPLM įrenginiuose tyrimas**

Autorius **Vladas Taluntis**

Vadovas **Tomyslav Sledevič**

Kalba: lietuvių

Anotacija

Magistro baigiamojo darbo tikslas – įgyvendinti duomenų kodavimo algoritmą lauku programuojamų loginių matricų įrenginiuose (LPLM), ištirti skirtingų modelių našumą ir kodavimo spartą, bei palyginti LPLM įtaisus su kitos rūšies procesoriais. Tiriamasis darbas suskirstytas į 3 etapus. Pirmajame etape apžvelgti kodavimo algoritmai, išanalizuoti tyrimo objektai. Sekančiame etape įgyvendintas kodavimo algoritmas LPLM, Raspberry pi ir STM įtaisuose, bei sukurtas dirbtinis neuronų tinklas, skirtas duomenų analizavimui. Paskutiniame etape pateikta pasirinktų įtaisų tyrimo rezultatai bei išvados. Darbo apimtis – 43 psl. teksto be priedų, 31 iliustr., 2 priedai.

Prasminiai žodžiai: Realus laiko šifravimas, AES, lauku programuojamų loginių matricų įrenginiai, Raspberry Pi, STM, telemetrija, vaizdo transliacija, dirbtinis neuronų tinklas.

Vilnius Gediminas Technical University
Faculty of Electronics
Department of Electronic Systems

ISBN ISSN
Copies No.
Date-.....-.....

Master Degree Studies **Electronics Engineering** study programme Master Graduation Thesis

Title **Investigation of the FPGA Based Telemetry Data Coding Algorithm Implementation**

Author **Vladas Taluntis**

Academic supervisor **Tomyslav Sledevič**

Thesis language: Lithuanian

Annotation

The aim of master's thesis is to implement data encoding algorithm in field programmable data array (FPGA) devices, analyze efficiency and encoding velocity of different models and compare FPGA devices with some other processors. The research is divided into 3 parts. In stage 1, encoding algorithms are reviewed and research objects are analyzed. Stage 2 involves implementation of encoding algorithm in FPGA, Raspberry pi and STM devices as well as creating of artificial neuron network for data analysis. Finally, in stage 3, analysis results of chosen devices are presented and findings are discussed. Thesis consists of: 43 p. text without appendixes, 31 figures, 2 appendixes included.

Keywords: Real-time encryption, AES, field programmable gate array devices, Raspberry Pi, STM, telemetry, video streaming, artificial neural network.

(Baigiamojo darbo sąžiningumo deklaracijos forma)

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

Vladas Taluntis, 20140174

(Studento vardas ir pavardė, studento pažymėjimo Nr.)

Elektronikos fakultetas

(Fakultetas)

Elektronikos inžinerija, EKSfm-15

(Studijų programa, akademinė grupė)

BAIGIAMOJO DARBO (PROJEKTO)

SĄŽININGUMO DEKLARACIJA

2017 m. gegužės 31 d.

Patvirtinu, kad mano baigiamasis darbas tema „Telemetrinių duomenų šifravimo algoritmo įgyvendinimų LPLM įrenginiuose tyrimas“ patvirtintas 2015 m. lapkričio 06 d. dekanų potvarkiu Nr. 218el, yra savarankiškai parašytas. Šiame darbe pateikta medžiaga nėra plagijuota. Tiesiogiai ar netiesiogiai panaudotos kitų šaltinių citatos pažymėtos literatūros nuorodose.

Prenkant ir įvertinant medžiagą bei rengiant baigiamąjį darbą, mane konsultavo mokslininkai ir specialistai: dr. Aušra Žemienė. Mano darbo vadovas dr. Tomyslav Sledevič.

Kitų asmenų indėlio į parengtą baigiamąjį darbą nėra. Jokių įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs (-usi).

(Parašas)

Vladas Taluntis
(Vardas ir pavardė)

TURINYS

SANTRUMPOS	9
1 ANALITINĖ LITERATŪROS APŽVALGA	12
1.1 Kodavimo algoritmų apžvalga	12
AES	12
DES	15
1.2 Kodavimo įrenginių apžvalga	16
Lauku programuojamos loginės matricos	16
Raspberry Pi 3 apžvalga	22
STM32F103 apžvalga	23
1.3 MAVLink protokolas	24
1.4 Dirbtinių neuronų tinklai	25
1.5 Telemetrija	27
2 DUOMENŲ ŠIFRAVIMO ALGORITMO KŪRIMAS IR ĮGYVENDINI-	29
MAS	
2.1 Kodavimo algoritmo įgyvendinimas	29
2.2 AES kodavimo įgyvendinimas XILINX Spartan-6	30
2.3 AES kodavimo įgyvendinimas STM32F103	33
2.4 Dirbtinių neuronų tinklo įgyvendinimas	34
3 DUOMENŲ ŠIFRAVIMO ALGORITMO TYRIMAS	39
3.1 Vaizdo duomenų kodavimas	40
4 APIBENDRINIMAS. IŠVADOS	42
LITERATŪROS SĄRAŠAS	43
PRIEDAI	44

SANTRUMPOS

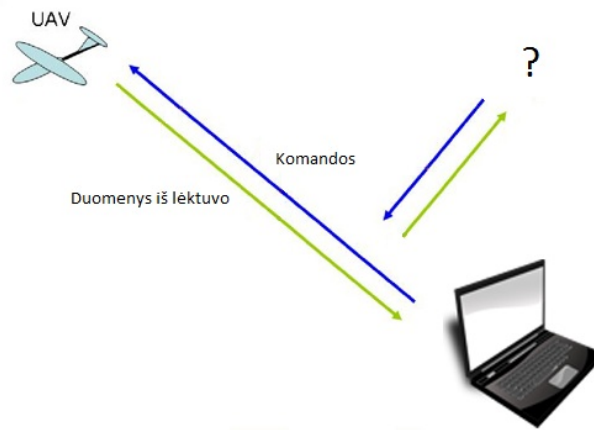
AES (angl. <i>Advanced Encryption Standard</i>)	pažangus kodavimo standartas
ASIC (angl. <i>Application Specific Integrated Circuit</i>)	specialios paskirties integrinis grandynas
CBC (angl. <i>Cipher Block Chaining</i>)	šifravimo blokų jungimas
CTS (angl. <i>Clear to Send</i>)	prašymas priimti
DES (angl. <i>Data Encryption Standard</i>)	duomenų kodavimo standartas
GPIO (angl. <i>General Purpose Input/Output</i>)	bendrosios paskirties įvestys/išvestys
FRB	funkciją vykdomasis blokas
HDMI (angl. <i>High Definition Multimedia Interface</i>)	aukštos raiškos multimedijos sąsaja
KAL	konfigūracinės atminties ląstelė
LUT (angl. <i>Look-Up Table</i>)	paieškos lentelė
LPLM	lauku programuojama loginė matrica
PLB	programuojamų lustų blokai
RISC (angl. <i>Reduced Instruction Set Computer</i>)	sumažintas kompiuterio komandų rinkinys
RTS (angl. <i>Request to Send</i>)	prašymas išsiųsti
SPI (angl. <i>Serial Peripheral Interface</i>)	serijinė periferinė sąsaja
SRAM (angl. <i>Static Random Access Memory</i>)	statinė operatyvioji atmintis
UART (angl. <i>Universal Asynchronous Receiver/Transmitter</i>)	universalus asinchroninis imtuvas/siųstuvas
USB (angl. <i>Universal Serial Bus</i>)	universalioji nuoseklioji jungtis
USART (angl. <i>Universal synchronous/asynchronous receiver/transmitter</i>)	universalus sinchroninis/asinchroninis imtuvas/siųstuvas
VHDL (angl. <i>Very High Speed Integrated Circuits Hardware Description Language</i>)	labai didelio greičio integrinių grandynų aparatūros aprašymo kalba
VGA (angl. <i>Video Graphics Array</i>)	vaizdo grafinė matrica

ĮVADAS

Magistro baigiamajame darbe nagrinėjamos lauku programuojamos loginės matricos (LPLM), kuriose įgyvendinamas kodavimo algoritmas. LPLM parenkama pagal duomenų srautą, kanalų skaičių, kodavimo algoritmo įgyvendinimo pobūdį, taip pat palyginama su „Raspberry Pi“ kompiuteriu ir „STM“ mikrovaldikliu.

Darbo aktualumas

Naudojant bepilotį orlaivį objektams stebėti, garsams įrašyti ar pramogai svarbu užtikrinti bepiločio orlaivio nenutrūkstamą valdymą ir perduodamų duomenų saugumą (1 pav.). Iš bepiločio orlaivio ir kompiuterio siunčiami signalai turi būti užkoduoti. Tam gali būti naudojamas „LPLM“ mikrograndynas arba kitokia įranga. Šiame darbe bus analizuojami galimi kodavimo įtaisai ir palyginami jų parametrai bei kodavimo greičiai.



1 pav. Duomenų pasisavinimas

Darbo tikslas – „LPLM“ įrenginyje įgyvendinti ir ištirti telemetrinių duomenų šifravimo algoritmą.

Darbo uždaviniai

- Atlikti analitinę literatūros apžvalgą, išanalizuoti duomenų kodavimo algoritmus bei jų įgyvendinimo galimybes;
- Sukurti kodavimo algoritmo programas, skirtas kodavimo įgyvendinimui LPLM, „Raspberry Pi“, „STM“ įrenginiuose;
- Atlikti sukurto algoritmo tyrimą su įvairiais „LPLM“ lustais ir gautų rezultatų analizę.

Darbo struktūra

Magistro baigiamojo darbo aiškinamąjį raštą sudaro 5 skyriai.

1 aiškinamojo rašto skyrius – įvadas.

2 skyriuje pateikiama analitinė literatūros apžvalga. Nagrinėjami šifravimo algoritmai, jų taikymo sritys, privalumai, trūkumai. Apžvelgiamos algoritmų įgyvendinimo galimybės LPLM, "Raspberry Pi", "STM" įrenginiuose.

3 skyrius skirtas algoritmų įgyvendinimui LPLM, "Raspberry Pi", "STM" įrenginiuose. Taip pat dirbtinių neuronų tinklo apmokymui. Pateikiami veikimo paaiškinimai.

4 skyriuje pateikiamas duomenų šifravimo algoritmo tyrimas. Tiriama LPLM, "Raspberry Pi" ir "STM" įtaisai, skaičiuojama kiekvieno įtaiso kodavimo/dekodavimo sparta, maksimali duomenų kodavimo/dekodavimo sparta, maksimalus įtaisų kodavimo kanalų kiekis.

5 skyriuje darbas apibendrinamas ir pateikiamos išvados apie duomenų šifravimo algoritmų įgyvendinimą LPLM, "Raspberry Pi" ir "STM" įrenginiuose.

1 ANALITINĖ LITERATŪROS APŽVALGA

1.1 Kodavimo algoritmų apžvalga

AES

AES (angl. Advanced Encryption Standard) – tai pažangus šifravimo standartas, kuriuo 2001 m. Nacionalinis standartų ir technologijų institutas (angl. National Institute for Standards and Technology, NIST) paskelbė pakeičiantis ankstesnį DES (Data Encryption Standard) kodavimą [1]. AES dar vadinamas „Rijndael“ algoritmu, jo autoriai – Joan Daemen ir Vincent Rijmen. „Rijndael“ yra blokinis simetrinis algoritmas, kurio bloko dydis yra 128 bitai. Rakto ilgiai gali būti 128, 192 arba 256 bitų. AES su 128 bitų raktu atlieka 10-ies ciklų, 192 bitų „AES“ 12-os ciklų, o 256 bitų raktas 14-os ciklų užkodavimą. AES suskaldo duomenis į 16-os baitų (128 bitų) blokus ir traktuoja kiekvieną bloką kaip 4×4 matricą, su kuria atlieka pakeitimus.

AES šifravimo etapai:

1. Baitų sumaišymas (SubBytes);
2. Eilučių perstūmimas (ShiftRows);
3. Stulpelių sumaišymas (MixColumns);
4. Užšifravimas (AddRoundKey);

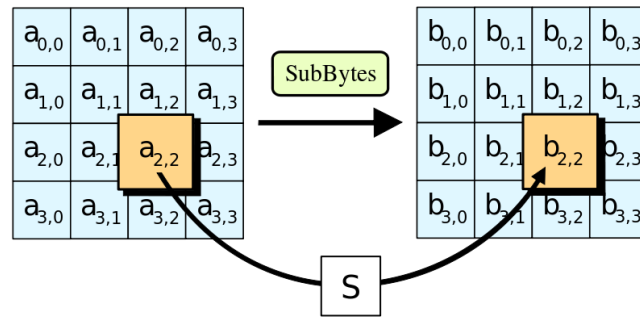
Baitų sumaišymas. Kiekvienas baitas yra apdorojamas pagal „S-Box“ (1 lentelė). „S-Box“ yra sukeitimo lentelė (angl. substitution table), kurioje vienas baitas pakeičiamas kitu, remiantis naudojamu algoritmu.

Lentlė 1. S-box duomenys

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1x	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2x	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3x	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4x	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5x	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6x	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7x	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8x	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9x	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
ax	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
bx	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
cx	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
dx	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
ex	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
fx	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

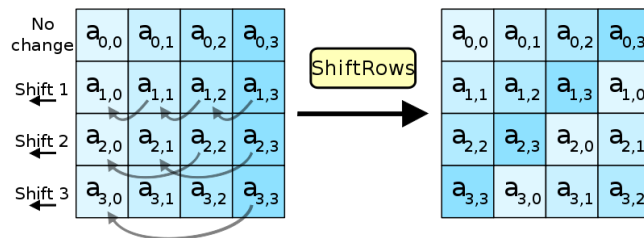
Norint atlikti baitų sumaišymo operaciją, pavyzdžiui, eilutėje „ABC“, reikia paimti kiekvieno baito šešiolyktinę vertę. ASCII „A“ == hex 0x41, „B“ == 0x42, o „C“ == 0x43. Pir-

masis (kairysis) skaitmuo atitinka lentelės eilutes, o antrasis stulpelius. 0x42 tampa 0x83; 0x43 tampa 0x2c, o 0x1a tampa 0x1b (2 pav.).



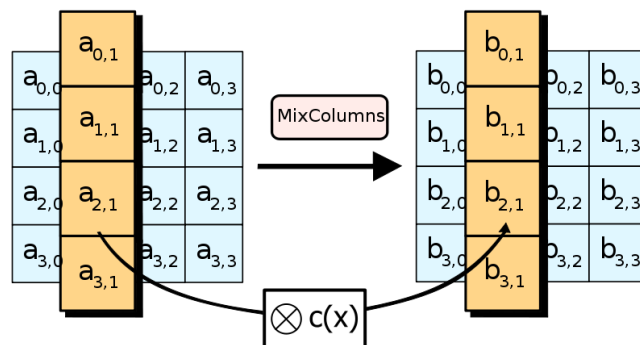
2 pav. Baitų sumaišymas

Eilučių perstumdyimas. Antruoju veiksmu „AES“ yra sumaišomos eilutės. Kiekvieno bloko nulinė eilutė nėra perstumdoma, pirmoji eilutė perstumdoma per vieną baitą, antroji per du, trečioji per tris baitus (3 pav.).



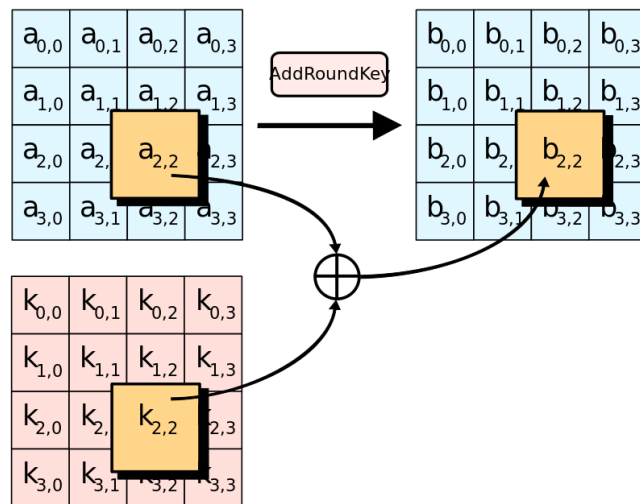
3 pav. Eilučių perstumdyimas

Stulpelių sumaišymas. Kiekvienas 4 baitų stulpelis yra traktuojamas kaip vienas 4 baitų ilgio numeris ir būna padauginamas iš matricos (4 pav.).



4 pav. Stulpelių sumaišymas

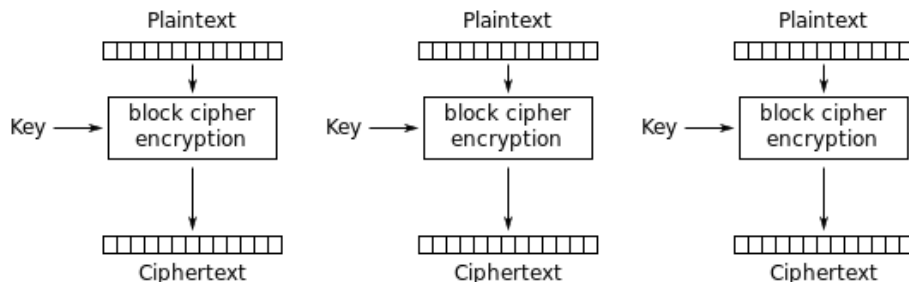
Užšifravimas. Bloko matrica sudedama su etapo rakto (Round key) matrica (5 pav.). Šios operacijos kartojamos 10 kartų (AES 128 atveju), tačiau dešimtą kartą neatliekama stulpelių maišymo operacija (MixColumns) [2].



5 pav. Užšifravimas

ECB algoritmas

Vienas iš paprasčiausių kodavimo algoritmų vadinamas elektronine kodų knyguote (ang. Electronic Codebook). Šiuo algoritmu informacija koduojama atkiris blokais, neturinčiais jokio tarpusavio ryšio (6 pav.). Šio kodavimo privalumas – nesudėtingas algoritmo įgyvendinimas, tačiau dėl to šifras yra labiau pažeidžiamas.

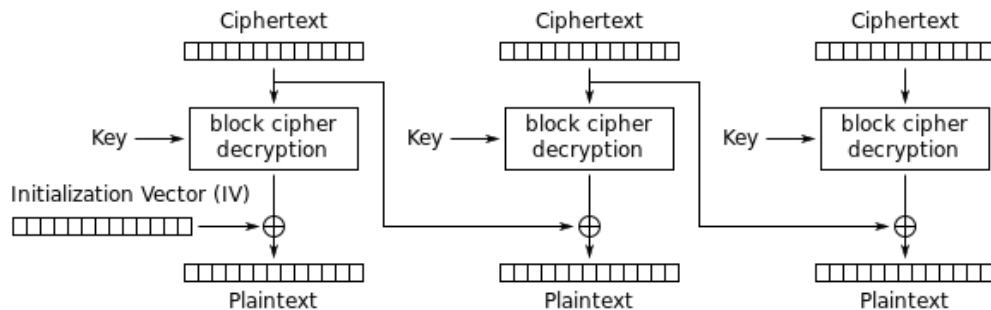


6 pav. ECB algoritmas

CBC algoritmas

CBC (angl. Cipher Block Chaining) algoritmą 1976m. sukūrė Ehrtman, Meyer, Smith ir Tuchman. Šis algoritmas prieš prasidedant naujam ciklui „XOR“ loginio elemento pagalba kiekvieną informacijos bloką palygina su prieš tai užkoduotu duomenų bloku. Taigi kiekvienas užkoduotas blokas yra priklausomas nuo prieš tai užkoduoto bloko (7 pav.). Norint, kad kiekvienas blokas būtų unikalus, pirmam blokui reikia naudoti inicializacijos vektorių. Inicializacijos vektorius neturi būti laikomas slaptu. Tai turėtų būti atsitiktinis skaičius (ar serijos numeris), kad būtų garantuota, jog kiekviena žinutė yra užšifruota unikaliu būdu. Klaida užšifruotam bloke (sukelta, pavyzdžiui, perdavimo nesėkmės) visiškai iškraipo bloką su klaida. Po jo sekantis blokas turės bitų klaidas tose pačiose vietose kaip ir originalus klaidingas blokas.

Blokai po antrojo bloko jau nebus paveikti klaidos. Vadinasi, CBC po bitų klaidų atsitaiso savaime, tačiau jis neatsitaiso po sinchronizacijos klaidų apskritai. Jei bitas yra pridėtas ar prarastas šifrogramos eilėje, tai visi vėliau einantys blokai bus iškraipyti. Sistema, kuri naudoja CBC, turi garantuoti, kad blokų struktūra lieka nesugadinta.



7 pav. CBC algoritmas

DES kodavimo apžvalga

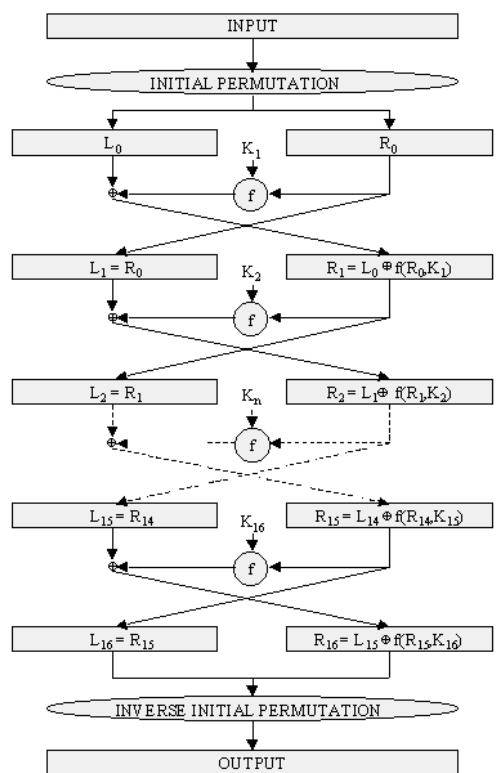
DES (angl. data encryption standart) – tai algoritmas, skirtas simetriniam duomenų šifravimui, sukurtas įmonės „IBM“ ir įsitvirtinęs kaip šifravimo standartas JAV. Bloko ilgis yra 64 bitai, lygiai perpus trumpesnis nei anksčiau minėtas AES blokas. Kodavimui naudojamas raktas, kurio ilgis – 64 bitai, bet 1 bitas kiekviename baite yra naudojamas kaip patikra (angl. parity), todėl lieka 56 bitai. Šifravimas vyksta blokais po 8 baitus, t.y., į DES kodavimo įvadą duomenys patenka dalimis po 8 baitus [3]. Prieš šifravimą vykdoma pirminė bitų permutacija, o baigus šifruoti – galutinė permutacija, atvirkštinė pirminei. Toliau vykdoma 16-kos analogiškų žingsnių seka, kai specialia „Feistelio“ (vienas iš kūrėjų) funkcija permutuojamos įvado dalys (2 po 32 bitus)(8 pav.) parenkant 48 bitų permutaciją iš rakto reikšmės. Šis kodavimas yra ganėtinai nesunkiai realizuojamas procesoriaus pagalba. Siekiant greitaveikos, permutacijos dalys bei L ir R porų keitimasis gali būti atliekami assembleriu. DES, kaip ir kitus permutacijos algoritmus, labai patogu realizuoti lauku programuojamos loginės matricos (LPLM) pagalba. Tačiau dėl rakto trumpumo „Copacabana“ mašina, kuri sukonstruota iš 120 LPLM, pilnai perskaičiuoja visus galimus rakto variantus nuo 6,4 iki 12,8 dienos.

- L_n, R_n – įvadas padalintas į 32 bitus.
- K_n – 48 bitų rakto dalis.

DES kodavimas turi keletą skirtingų algoritmų, dėl kurių šifruoto teksto nulaužimas tampa dar sudėtingesnis:

- ECB (angl. electronic code book) – elektroninės kodinės knygos algoritmas.
- CBC (angl. cipher block chaining) – specialių blokų algoritmas.
- CFB (angl. cipher feed back) – atgalinio ryšio algoritmas pagal šifravimo tekstą.

- OFB (angl. output feed back) – atgalinio ryšio algoritmas pagal šifrą.



8 pav. Feistelio funkcija

1.2 Kodavimo įrenginių apžvalga

Lauku programuojamos loginės matricos

„LPLM“ buvo išvystyta iš programuojamų pastovios atminties įtaisų (angl. ROM) ir programuojamų loginių schemų (angl. PLD). Abu šie prietaisų tipai yra programuojami; tai gali padaryti gamintojas arba vartotojas. Taigi įrašyta programa nustato LPLM tranzistorių būseną (atidarytas arba uždarytas) ir taip sudaro loginį grandyną.

Istoriškai LPLM buvo lėtesnės, naudodavo mažiau energijos ir turėdavo mažiau funkcijų nei specialios paskirties integrinis grandynas (angl. Application Specific Integrated Circuit (ASIC)). Lyginant su ASIC, dizaino įgyvendinimas LPLM reikalauja vidutiniškai 40 kartų daugiau ploto ir veikia viena trečiąja ASIC greičio, tačiau šiuos trūkumus atperka galimybė perprogramuoti ir ištaisyti klaidas. Pirmą komercinę lauku programuojama loginė matrica buvo pristatyta 1984 m. JAV elektronikos inžinierių Ross'o Freeman'o ir Bernard'o Vonderschmitt'o iš „Xilinx“ įmonės. Jie pristatė pirmąjį komercinį „XC2064“ modelį (9 pav.), kuris turėjo 64 konfigūruojamus loginius blokus (angl. CLBs) ir 58 įvadus ir išvadus [4]. Po maždaug 20 metų Ross Freeman pateko į nacionalinę šlovės alėją už LPLM sukūrimą.

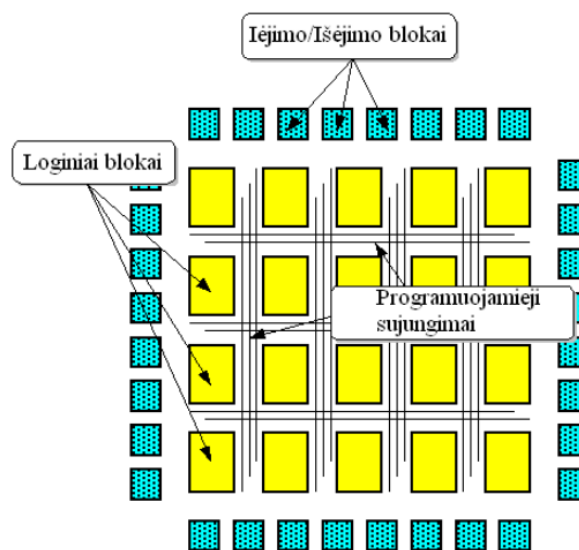


9 pav. Pirmas komercinis LPLM modelis

Lyginant su pirmaisiais LPLM, vėlesnieji gerokai pranašesni: dabartinius LPLM vidutiniškai sudaro 330000 loginių blokų ir 1100 įvadų bei išvadų.

LPLM sandara

LPLM – tai puslaidininkių elementų (pvz., trigerių, multiplekserių ar sumatorių), tarpusavyje sujungiamų perprogramuojamais loginiais ryšiais, laukas (10 pav.). Šios savybės išskiria LPLM iš kitų įterptinių sistemų kaip sistemą, leidžiančią įgyvendinti įvairaus tipo uždavinius, kuriant kompleksinius loginius grandynus. Loginėms sąsajoms tarp vidinių LPLM blokų sudaryti bei grandynams įgyvendinti skirtos techninio lygmens programavimo kalbos VHDL ir „Verilog“. LPLM plačiai naudojamos prireikus atlikti specifinius uždavinius ir norint patobulinti ir paspartinti sistemos veikimą. Šifruojant duomenis realiu laiku galima naudoti ASIC procesorius, kurie gaminami specifinėms užduotims atlikti, tačiau jų gamybos sąnaudos yra didesnės nei „LPLM“ procesorių ir jų nebegalima pritaikyti kitai funkcijai atlikti. Tuo tarpu LPLM leidžia projektuotojams nebrangiai modeliuoti, kurti schemas ir jas įgyvendinti. Toliau aprašomas šiame darbe naudotas LPLM įrenginys.

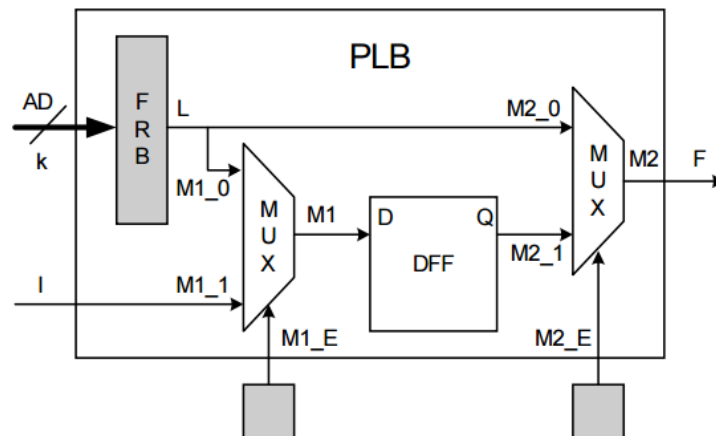


10 pav. Lauku programuojamos loginės matricos struktūra

Programuojamų lustų architektūra

Paprastai kiekviena programuojamųjų lustų šeima pasižymi specifinėmis savybėmis. Dažniausiai tokie lustai būna sudaryti iš loginių blokų [5], t.y., „PLM“, kurių dydis $N \times M$ (dažniausiai N ir M būna lygūs), bei programuojamųjų įvadų ir išvadų blokų. Visa tai būna sujungta programuojamuoju ryšių tinklu. Nagrinėjant SRAM technologijos lustus, kurie programuojami į lusto konfigūracinės atminties ląsteles (KAL) įrašant reikiamas vertes. Iš esmės visų tipų programuojamųjų lustų blokai (PLB) savo vidine struktūra yra panašūs (11 pav.). Pilkos spalvos blokai žymi konfigūracinės atminties ląsteles. Loginiai blokai paprastai būna sudaryti iš trijų pagrindinių dalių:

- funkciją vykdančio bloko (FRB);
- multiplekserių;
- vieno D tipo trigerio.



11 pav. Programuojamosios logikos bloko (PLB) modelis

Funkciją vykdomasis blokas gali būti užprogramuotas taip, kad atliktų k įvadų turinčią kombinacinę funkciją arba būtų panaudotas kaip 2^k bitų RAM atmintis. Kuriant kombinacinę logiką, į konfigūracinę atmintį įrašoma 2^k eilučių turinti teisingumo lentelė; čia k yra FRB įvadų skaičius. Priklausomai nuo FRB įvaduose esančios vertės išrenkamas konfigūracinės atminties adresas ir išvade suformuojama tuo adresu saugoma atminties vertė, t.y. gaunamas funkcijos rezultatas. Taigi, FRB gali vykdyti bet kurią kombinacinę funkciją, turinčią ne daugiau kaip k argumentų. Programuojant lustus į FRB atstojančią konfigūracinę atmintį įrašomos atliekamos funkcijos bitų sekos, atitinkančios teisingumo lentelę. PLB vidiniai sujungimai tarp FRB ir „D“ trigerio taip pat kontroliuojami atitinkamomis KAL. PLB supanti ir jungianti sujungimų struktūra sudaryta iš tranzistorių, kurie taip pat kontroliuojami per KAL.

Apskritai lustai programuojami į jų konfigūracinę atmintį įkraunant reikiamą programą, aprašytą kaip bitų seka. Norint išsaugoti kiekvieną tokios programos bitą, į atitinkamą konfigūracinės atminties ląstelę konfigūruojami PLB, lusto įvadų ir išvadų blokai ir ryšiai tarp jų. Taip užprogramuotas lustas vykdo jam skirtą funkciją, kuri paprastai vadinama konfigūracija.

LPLM mikrograndynų apžvalga

Lauku programuojamų loginių matricių mikrograndynai būna įvairių tipų; kiekvienai užduočiai parenkami atitinkami grandynai. Akivaizdu, kad pasirinkus mikrograndyną su geriausiais parametrais nekiltų jokių problemų kuriant ar tobulinant sistemą, tačiau dažnu atveju būtų neišnaudojama didelė dalis loginių blokų ar skaitmeninio signalo procesorių. Todėl tokio mikrograndyno kaina neatitiktų vykdomos užduoties. Žemiau pateikiama iš penkių skirtingų LPLM mikrograndynų šeimų sudaryta lentelė, kurioje pateikti pagrindiniai parametrai (2 lentelė).

Lentelė 2. Pagrindiniai mikrograndynų parametrai

Device	Logic cells	Total RAM bits	Max user I/O	DSP	Max. Frequency
XC3S200	4 320	294 912	76	x	320MHz
XC6SLX16	14 579	589 824	232	32	330MHz
XC4VFX12	279	700 416	320	32	400MHz
XC5VLX20T	1591	958 464	172	24	400MHz
XC7VX330T	3840	27 648 000	700	1 120	1818MHz

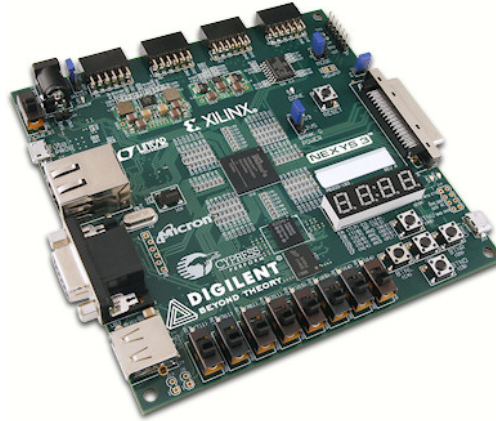
Kadangi realūs bandymai su visais mikrograndynais yra sunkiai įgyvendinami, buvo pasirinktas vienas iš lentelėje esančių mikrograndynų ir įgyvendintas realus duomenų kodavimo algoritmas su tikrais duomenimis. Su visais likusiais mikrograndymais LPLM algoritmas atliktas virtualioje aplinkoje, kuri bus aprašyta vėlesniame skyriuje. Pasirinkau „Spartan-6“ šeimos mikrograndyną, kuris yra „DIGILENT Nexys 3“ (12 pav.) bandymų modulyje. Ši plokštė skirta aukštosios mokykloms ir studentams atlikti įvairius bandymus ir tyrimus, joje daug išvadų, kuriais galima prijungti išorinius įtaisus. Trumpas pasirinkto mikrograndyno aprašymas:

- 14579 loginių ląstelių;
- 576Kb atminties;
- 232 išvadų ir išvadų;
- 18224 trigerių.

Bandymų modulyje taip pat yra 16 Mb operatyviosios atminties RAM (angl. Random Access Memory), SPI (angl. Serial Peripheral Interface). Darbui su šiuo mikrograndynu yra integruoti šie prievadai bei jungtys:

- 8-bit VGA išvadas;
- 10/100 Ethernet;
- USB-USART;
- 4 septynių segmentų indikatoriai;

- 5 mygtukai;
- 8 jungiklių;
- 8 žalios spalvos šviesos diodų.



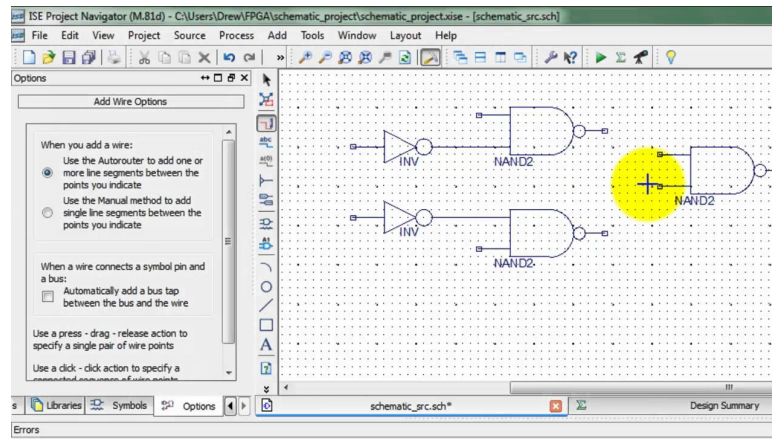
12 pav. „Nexys“ 3 bandymų plokštės bendras vaizdas

LPLM programavimas

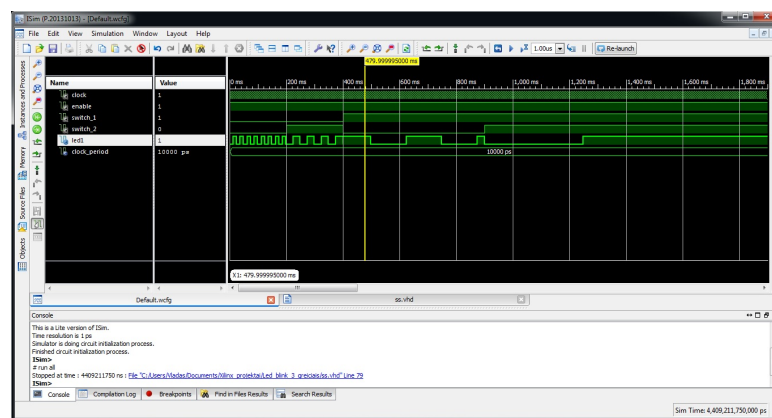
LPLM lustams programuoti kompanija „Xilinx“ siūlo „ISE Design Suite“ programą. Įdiegę programą į kompiuterį ir prie jo per USB (angl. Universal Serial Bus) prievadą prijungiamo bandymų plokštę. Prireikus įdiegiame programatoriaus tvarkyklę.

Loginių elementų realizavimas gali būti vykdomas dvejopai: rašomas programos kodas arba formuojama elektrinė principinė schema. Į klausimą, kokį metodą geriausiai naudoti, viena-reikšmiškai atsakyti sudėtinga, kadangi abu metodai turi savo privalumų ir trūkumų. Dėl to projektuojant geriausia naudoti abu metodus. Pasaulyje plačiai naudojamos dvi programavimo kalbos – tai VHDL ir „Verilog“. „ISE Design“ programinė įranga supranta bet kurią „HDL“ kalbą [6]. „ISE Design“ programa formuojant elektrinę principinę schemą viskas iš pradžių atrodo ganėtinai paprasta (13 pav.), tačiau kuriant sudėtingesnę schemą arba programą geriau naudoti VHDL kalbą, nes braižant viską schematiškai galima labai greitai pasimesti. Be to, sudėtingos schemas dažniausiai atrodo labai painios ir nesuprantamos.

Vienas iš pagrindinių šios programos privalumų yra galimybė simuliuoti sudarytą schemą arba parašytą kodą VHDL ar „Verilog“ kalba. Norint simuliuoti programą projekte reikia susikurti naują failą, kuriame bus aprašomi simuliacijos duomenys ir simuliacijos scenarijus. Ši galimybė sutaupo daug laiko, nes nereikia generuoti ir įkelti specialaus programos failo į LPLM. Be to, pagal laiko diagramas (14 pav.) galima sekti programos veikimą ir jos teisingumą.



13 pav. ISE Design programos schemos braižymo pavyzdys



14 pav. ISim programos simuliavimo langas

VHDL

VHDL – tai skaitmeninės elektroninės aparatūros funkcionavimo aprašymo kalba. Aparatūros aprašymo kalbų buvo sukurta labai daug. VHDL yra aparatūros aprašymo tarptautinis standartas, kuris leidžia aprašyti bet kokią aparatūriškai realizuojamą algoritmą, todėl VHDL kalba yra labai panaši į programavimo kalbą. Ši kalba šalia konstantų ir kintamųjų dar turi signalus, su kuriais susiejami laiko parametrai ir galima nurodyti operatorių vėlinimus. Be to, ji leidžia aprašyti procesus, kurie vykdomi lygiagrečiai. Šia kalba aprašytas įrenginys gali būti modeliuojamas kompiuteryje ir jo veikimas analizuojamas pagal laikines diagramas, panašiai kaip su oscilografu. Pagrindinis privalumas yra tas, kad šia kalba aprašytas įrenginys vėliau gali būti automatiškai sintezuojamas. Automatinės sintezės priemonės užtikrina, kad susintezuota schema veiktų taip pat kaip ir aprašytas įrenginys. Todėl elektroninio įrenginio aprašas pakeičia maketą ir projektas gali būti demonstruojamas VHDL modeliavimo priemonėmis. Šiuo metu jau yra sukurtos programinės priemonės, leidžiančios pilnai modeliuoti ir gauti laikines diagramas bet kokiam „VHDL“ kalbos aprašui [7]. Laiko parametrai įgalina naudoti VHDL kalbą ne tik aparatūros aprašymui, bet ir aprašant bet kokias realaus laiko sistemas. VHDL gali būti naudojama kaip imitacinio modeliavimo kalba sudėtingoms sistemoms, aprašomoms stochastinėmis priklausomybėmis. VHDL gali būti ir vykdomoji specifikavimo kalba. Šios kal-

bos pagalba galima modeliuoti sistemas, aprašomas diferencinėmis lygtimis. Ir visa tai palaiko grafinė įėjimo poveikių įvedimo ir rezultatų išvedimo sąsaja. Dėl šių priežasčių VHDL kalba itin paplitusi ir reikšminga tarp elektroninės aparatūros projektuotojų.

Verilog

„Verilog“ kalba buvo pradėta naudoti „Gateway Design Automation“ kompanijoje 1984 m., perimant struktūrą iš „HiLo“ ir „C“ kalbų, bet oficialiai ji pripažinta 1995 m. [8]. Ši kalba sukurta „C++“ kalbos pagrindu ir, kaip ir VHDL, gali būti tiek struktūrinė, tiek elgsenos.

Elgsenos specifikacija apibūdina skaitmeninės sistemos (modulio) elgseną naudojant įprastas programavimo kalbos konstrukcijas, pvz.: `is`, `case`.

Struktūrinė specifikacija apibūdina skaitmeninės sistemos (modulio) elgseną kaip hierarchinę sistemą tarp mažesniojo lygmens modulių. Žemiausiame lygmenyje komponentai turi būti primityvūs arba specifiukuota jų elgsena. Pagrindinis sudaromasis blokas, kaip ir VHDL kalbos atveju, yra modulis. Kiekvienas modulis turi sąsają su kitais moduliais. Moduliai gali veikti lygiagrečiai, bet įprastai būna tik vienas aukščiausiojo lygmens modulis, kuriame paskelbiami visi sudarantys sistemą modeliai. Verilog elgsenos lygmuo suskirstytas ir veikia lygiai taip pat kaip „C“ programavimo kalba. Elgsenos lygmens struktūros algoritminiai bei „TPL“ modeliai:

- struktūrinės procedūros nuosekliai arba lygiagrečiai vykdymui;
- aiški procedūrų kontrolė aktyvacijai, specifiukuotai pagal vėlinimus arba reikšmių pasikeitimo įvykius;
- kontroliuojami įvykiai aktyvuojant arba panaikinant tam tikrus veiksmus procedūrose;
- sąlyginiai sakiniai, ciklai;
- procedūros, funkcijos;
- aritmetinės, loginės, bitų operacijos.

Raspberry Pi 3 apžvalga

„Raspberry Pi 3“ – tai galingas mini kompiuteris. Pirmas tokio tipo kompiuteris išleistas 2012 m. Jungtinėje Karalystėje. Jame įtaisytas keturių branduolių „ARM Cortex-A53“ procesorius, kuris yra paremtas „ARMv8-A“ (64/32-bit) architektūra. „Raspberry Pi“ veikimas paremtas sisteminio lusto technologija (angl. System on a chip, SOC), kai didžioji dalis elektronikos (kartu su procesoriumi) yra suprojektuota į vieną lustą. Pagrindiniai parametrai:

- 1,2 GHz procesoriaus taktinis dažnis;
- 1 GB operatyviosios atminties;
- 4 USB lizdai;

- 40 GPIO prievadų.

Modulyje taip pat yra „etherneto“, HDMI ir garso jungtys, integruoti „wifi“ ir „bluetooth“, moduliai [9]. Šiame kompiuteriye yra idiegta „Linux“ operacine sistema „Raspbian“ kuri yra sukurta „Unix“ pagrindu. GPIO prievadai gali būti naudojami kaip loginiai įvestys ar išvestys. Taip pat dalis jų gali būti prijungti prie duomenų perdavimo kanalų (15pav.), pvz.: „I2C“, „SPI“, „UART“. UART išėjimo kanalo prievadai pažymėti kaip „RXD0“ ir „TXD0“. Pastebima, kad „Raspberry Pi 3“ palaiko tik vieną UART kanalą naudodamas GPIO prievadus. Todėl bus naudojamas kitas būdas įvesti ir išvesti duomenų srautui per UART sąsają. Internetinėje prekyboje plačiai paplitę „USB-UART“ keitikliai, kurie naudoja „FT232RL“ valdiklį. Juos galima pritaikyti „Raspberry Pi 3“ nes jie jungiami tiesiai į „USB A“ tipo jungtį.

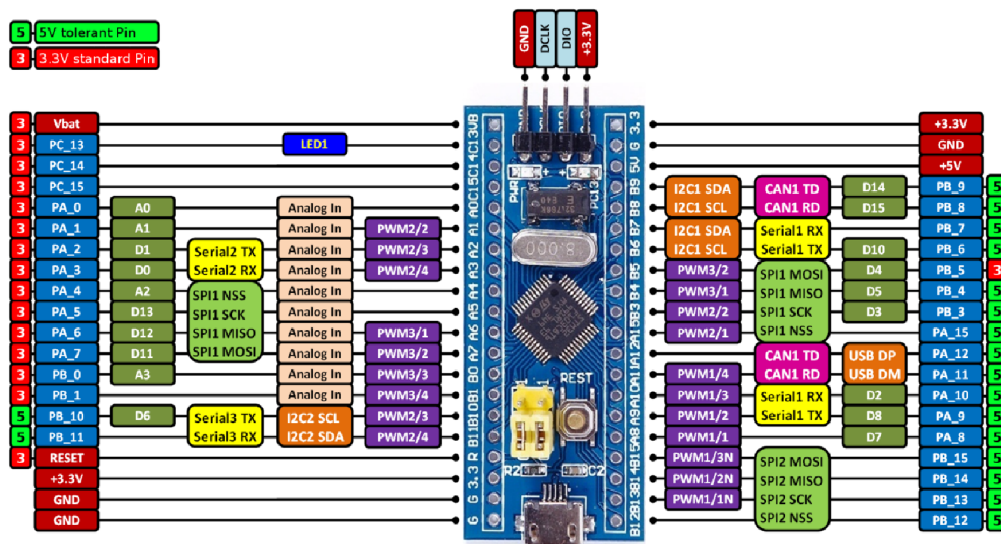
Raspberry Pi 3 GPIO Header				
Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1, I ² C)		DC Power 5v	04
05	GPIO03 (SCL1, I ² C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)		(I ² C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

15 pav. „Raspberry Pi 3“ B GPIO prievadų reikšmės

STM32F103 apžvalga

Mikrovaldiklis „STM32F103“ yra nedidelių matmenų modulis, kurį sudaro „ARM Cortex-M3“ 32-bit RISC architektūros procesorius [10] (didžiausias veikimo dažnis yra 72MHz). Taip pat šiame modulyje yra:

- 64 KB pastovios flash atminties;
- 20 KB operatyvioji atmintis;
- 3×USART moduliai, 2×I2C, 2×SPI sąsajų moduliai;
- 37 vartotojui naudojami išvadai.

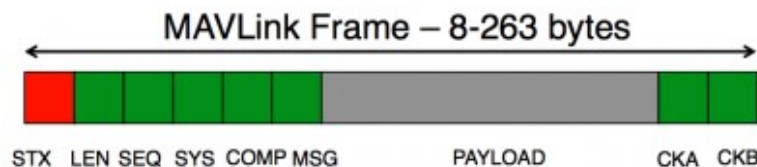


16 pav. STM32f103 įvadų ir išvadų žemėlapis

Norint valdyti šį mikrovaldiklį reikia specialaus kompiliatoriaus, kuris pakeistų tam tikrą programavimo kalbą parašytą kodą į mikrovaldikliui suprantamą kalbą. Geriausiai tinkanti programinė įranga yra „Keil uVision5“, kurioje bus rašomas kodavimo algoritmas „C“ kalba.

1.3 MAVLink protokolas

Protokolas, kuriuo perduodami duomenys – „MAVLink“ (17 pav.). Jis buvo sukurtas pagal „CAN“ ir „SAE AS-4“ standartus [11].



17 pav. MAVLink protokolo paketas

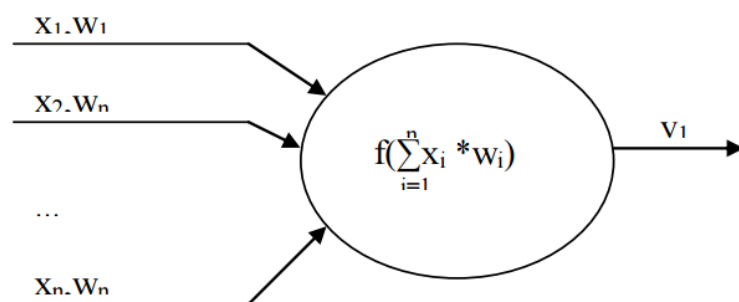
- STX – pradžios baitas (0xFE). Nurodo paketo pradžią;
- LEN – duomenų (angl. payload) ilgis. Nuo 0 iki 255;
- SEQ – komponentų skaitiklis. Leidžia aptikti prarastus paketus. Nuo 1 iki 255;
- SYS – sistemos identifikatorius. Leidžia kreiptis į atskiras „MAV“ sistemas. Nuo 0 iki 255;
- COMP – komponentų identifikatorius. Leidžia kreiptis į atskirus objektus „MAV“ tinkle. Nuo 0 iki 255;
- MSG – žinutės „ID“ nurodo, kokia informacija yra perduodama duomenų skiltyje ir kaip ji turi būti dekoduojama. Nuo 0 iki 255;

- PAYLOAD – žinutės duomenys, kurie priklauso dar ir nuo žinutės „ID“ (MSG). Nuo 0 iki 255 baitų;
- CKA,CKB – sumos tikrinimas, tikrinama, ar neprarasta duomenų perdavimo metu. Jei prarasta, su algoritmu jie gali būti atkurti.

1.4 Dirbtinių neuronų tinklai

Dirbtinis neuroninis tinklas yra informacijos apdorojimui skirta struktūra, sudaryta iš sujungtų, skaičiavimus atliekančių struktūrinių vienetų, vadinamų neuronais. Dirbtinį neuroną galima apibūdinti kaip tam tikrą funkciją, atitinkančią biologinio neurono modelį [12]. Kiekvienas neuronas turi keletą įvesčių ir vieną išvestį. Neuroninis tinklas geba spręsti uždavinius remdamasis jam prieinamais pavyzdžiais. Tinklo apmokymo režime pagal turimus pavyzdžius neuronas keičia su įvestimis susietų svorių reikšmes taip, kad būtų gauta pageidaujama išvesties reikšmė. Pastaruosius svarbu kruopščiai parinkti, kadangi netinkamus pavyzdžius tinklas gali pernelyg ilgai analizuoti arba išvis suprasti nekorektiškai. Naudojimo režime neuronas grąžina su įvestyse esančiomis reikšmėmis susijusią reikšmę. Neuroniniams tinklams būdingos kelios pagrindinės savybės:

- Prisitaikymas ir mokymasis suteikia galingas ir efektyvias apdorojimo galimybes;
- Netiesinis tinklo apdorojimas padidina tinklo apibendrinimo laipsnį, klasifikacijos ir triukšmo pašalinimo galimybes;
- Lygiagretūs skaičiavimai, atliekami tarpusavyje susietų ląstelių;
- Klaidų tolerancija – sugedus daliai tinklo nukenčia našumas bei tikslumas, tačiau tam tikros tinklo funkcijos vis tiek vykdomos patikimai.



18 pav. Neurono modelis

Perceptronas

Perceptronas – tai neurono su įvestimis modelis, papildytas tam tikromis papildomomis pradinio apdorojimo galimybėmis. Prieš analizuojant įvesties duomenys apdorojami ir iš jų išgaunamos tam tikros savybės. Tada tolesni skaičiavimai vykdomi su naujomis savybėmis, užuot naudojus visus pradinius duomenis.

Tiesioginio sklidimo tinklai

Tiesioginio sklidimo neuroniniuose tinkluose signalai sklinda tik viena kryptimi, t.y., tolimesnio sluoksnio neuronų išvestys gauna informaciją tik iš prieš tai buvusių sluoksnių. Tokiuose tinkluose nėra nei grįžtamųjų ryšių iš ankstesnių sluoksnių, nei ciklų su tame pačiame sluoksnyje esančiais neuronais. Prie tiesioginio sklidimo neuroninių tinklų priskiriami:

- Viensluoksnis perceptronas – kai tarp įvesties ir išvesties sluoksnių nėra paslėptų neuronų sluoksnių;
- Daugiasluoksnis perceptronas – kai tarp įvesties ir išvesties sluoksnių yra vienas ar daugiau paslėptų neuronų sluoksnių;
- Radialinių bazinių funkcijų tinklai – kai paslėptuose sluoksniuose kaip aktyvavimo funkcija yra naudojamos radialinės bazinės funkcijos.

Tinklo sluoksniai

Galima būtų išskirti vienasluoksnius ir daugiasluoksnius neuroninių tinklų modelius. Vienasluoksnės architektūros atveju visi neuronai yra tarpusavyje susiję, kai tuo tarpu daugiasluoksnės architektūros atveju neuronai yra hierarchiškai suskirstyti į sluoksnius, kur kiekvieno sluoksnio neuronų įvestys susijusios tik su prieš tai buvusio sluoksnio neuronais. Dažniausiai neuroniniai tinklai būna sudaryti iš trijų sluoksnių grupių:

Įvesties sluoksnio neuronai yra atsakingi už duomenų įvedimą į tinklą. Kartais jie taip pat vykdo tam tikrą pradinį įvesties duomenų apdorojimą. Pavyzdžiui, vykdant rašmenų atpažinimą gali tekti suvienodinti visų simbolių aukščius, nes skirtingų žmonių rašysenos yra skirtingos.

Paslėptųjų sluoksnių neuronai remdamiesi įvesties reikšmėmis ir apmokymo metu nustatytais jungčių svoriais vykdo su uždavinio sprendimu susijusius skaičiavimus.

Išvesties sluoksnio neuronų veikla priklauso nuo paslėptųjų sluoksnių neuronų išvesties reikšmių ir su jais susijusių ryšių svorių.

Tiesioginio sklidimo tinklai

Tiesioginio sklidimo neuroniniuose tinkluose signalai sklinda tik viena kryptimi, t.y., tolimesnio sluoksnio neuronų išvestys gauna informaciją tik iš prieš tai buvusių sluoksnių. Tokiuose tinkluose nėra nei grįžtamųjų ryšių iš ankstesnių sluoksnių, nei ciklų su tame pačiame sluoksnyje esančiais neuronais. Prie tiesioginio sklidimo neuroninių tinklų priskiriami:

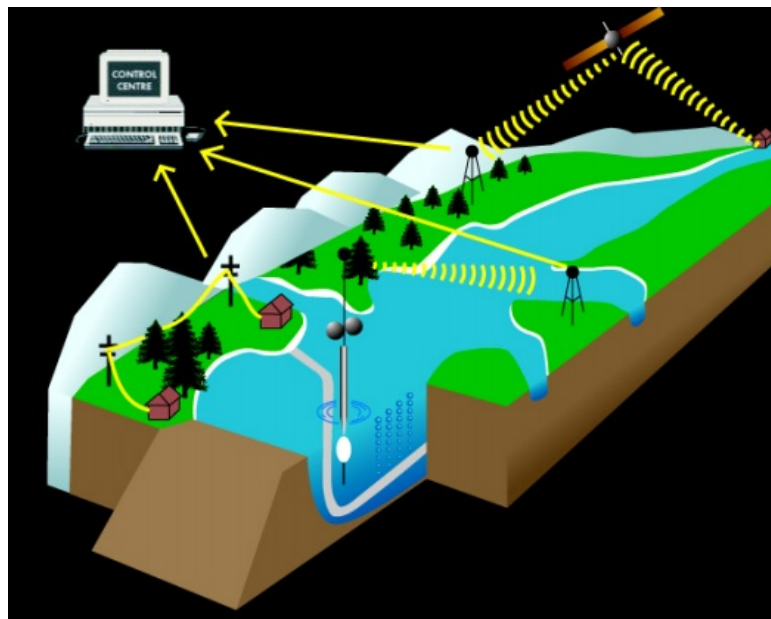
- Viensluoksnis perceptronas – kai tarp įvesties ir išvesties sluoksnių nėra paslėptų neuronų sluoksnių;
- Daugiasluoksnis perceptronas – kai tarp įvesties ir išvesties sluoksnių yra vienas ar daugiau paslėptų neuronų sluoksnių;

- Radialinių bazinių funkcijų tinklai – kai paslėptuose sluoksniuose kaip aktyvavimo funkcija yra naudojamos radialinės bazinės funkcijos.

1.5 Telemetrija

Telemetrija – tai automatinis susisieikimo procesas, kurio metu gaunama informacija iš nutolusių matavimo prietaisų. Šiuo tikslu naudojami radijo, mobiliųjų telefonų bei kitokie bevielio susisieikimo ryšiai. Telemetrijos pagalba galima sužinoti apie kelių įtaisų parametrus, kurie dažniausiai būna nutolę vienas nuo kito [13]. Tokios sistemos pavyzdys: centrinis kompiuteris gauna duomenis iš skirtingų vandens bokštų apie vandens kiekį ar naudojamą elektrą (19 pav.). Pirmieji telemetriniai procesai buvo atliekami XIX amžiuje, susisieikimui naudojant laidus. Laidai naudojami iki šiol, tačiau tai nėra patogus būdas. Šiais laikais dažniausiai naudojami bevieliai tinklai, tačiau viskas priklauso nuo to, kiek prietaisai atitolę nuo stebėtojo. Galima išskirti keletą telemetrijos panaudojimo būdų:

- Reikiamu momentu siunčiant signalą iš centrinio kompiuterio į įtaisą, kuris gali sužinoti įtaiso nustatytas reikšmes tik tuo momentu;
- Galima pasirinkti vieną ar daugiau įtaisų ir iš jų gauti duomenis;
- Įtaisų duomenis stebėti nuolat atnaujinant informaciją.



19 pav. Telemetrijos veikimo principo pavyzdys

Bevielio susisieikimo tipai

Radio telemetrija

Radio ryšio naudojimas telemetriniams procesams turi keletą privalumų ir trūkumų. Visu pirma, tokia sistema nėra brangi. Dažniausiai naudojamas 2,4 GHz dažnis, naudojant šį

dažnį nereikalingos licenzijos, tačiau jo veikimo diapazonas pakankamai nedidelis (iki 15 km).

Mobili telemetrija

Ši telemetrijos komunikavimo sistema naudoja mobilių ryši. Ji nebrangi ir nesunkiai įdiegiama, tačiau naudojant mobilių ryši už ją reikia mokėti pinigų, užtikrinti gerą ryši, nes kitoku atveju galima prarasti informaciją. Ryšys gali nutrūkti dėl linijos perkrovos piko metu.

Palydovine telemetrija

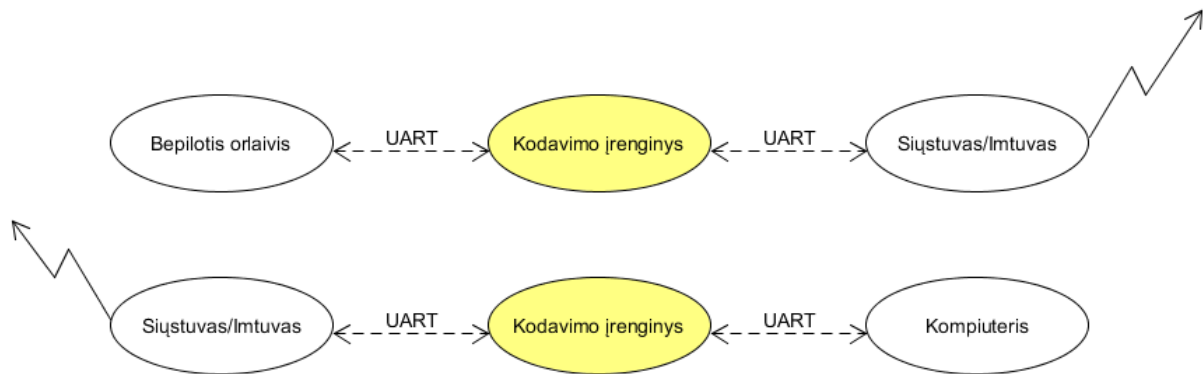
Ši sistema geriausiai tinka:

- Vietoms, kuriose nėra mobilaus ryšio;
- Tolimiems taškams, kurių nepasiekia radijo dažnis.

Tačiau nevertėtų pamiršti, kad norint naudoti palydovinį ryši reikia gauti leidimą.

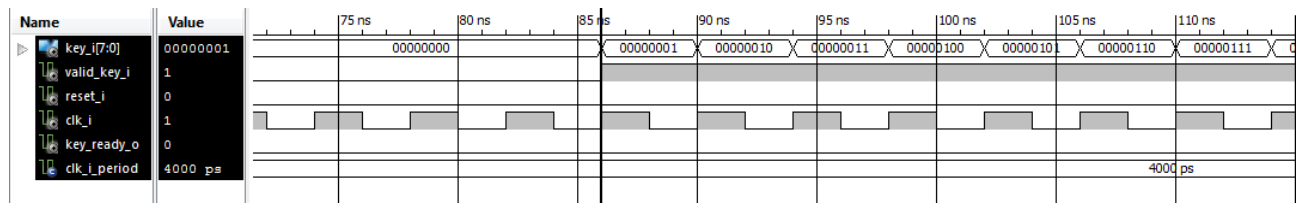
2 DUOMENŲ ŠIFRAVIMO ALGORITMO KŪRIMAS IR ĮGYVENDINIMAS

2.1 Kodavimo algoritmo įgyvendinimas



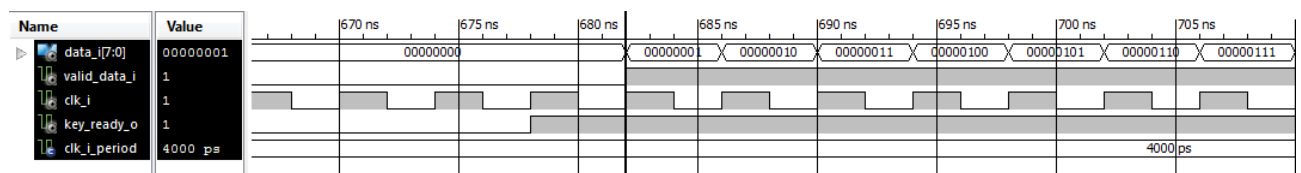
20 pav. Raspberry Pi telemetrinių duomenų kodavimo įgyvendinimas

Pagrindinė užduotis – telemetrinius duomenis, siunčiamus tarp kompiuterio ir bepiločio lėktuvo, užkoduoti naudojant AES kodavimo algoritmą (20 pav.). Duomenų srautas perduodamas UART sąsaja naudojant „MAVLink“ protokolą. ISim aplinkoje paleidus kodavimo algoritmą laukiama „reset_i“ būsenos iš 0 į 1 pakitimo. Kai tai įvyksta, signalo „key_ready_o“ būsena pasikeičia į 0, tuomet į kodavimo algoritmą įvedamas kodavimo raktas, sudarytas iš 16 baitų (21 pav.).



21 pav. Kodavimo rakto įvedimas

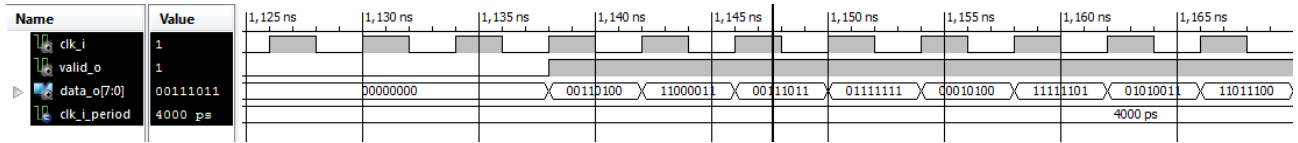
Kai įvestas raktas baigiamas apdoroti, „key_ready_o“ reikšmė pasikeičia į 1. Praėjus vienam taktinio dažnio periodui (angl. clock), trunkančiam 4 ns, pradedamas duomenų įvedimas signalu „data_i“ (22 pav.).



22 pav. Duomenų įvedimas

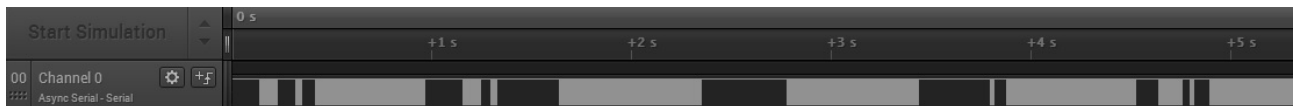
Duomenų įvedimo pabaigoje signalo „valid_data_i“ būsena pasikeičia į 0 ir pradedamas 12 ciklų duomenų šifravimas. Įvesti duomenys bus šifruojami (jų dydis – 16 baitų). Kai

kodavimo algoritmas įvykdomas, signalo „valid_o“ būseną pasikeičia į 1, signalo „data_o“ išvestyje gauname užšifruotus duomenis (23 pav.).



23 pav. Užkoduotu duomenų išvedimas

Koduojant duomenys turi būti suskirstyti į vienodo dydžio blokus (16 baitų). Išanalizavus išmatuotą duomenų srautą (24 pav.), perduodamą UART sąsaja, pastebima, kad bepiločio orlaivio siunčiami arba priimami duomenys ne visuomet būna vienodo dydžio. Šiai problemai išspręsti reikia išanalizuoti protokolą, kuriuo yra perduodami duomenys, ir jį pritaikyti kiekvienam įtaisui individualiai.



24 pav. UART duomenų srautas.

2.2 AES kodavimo įgyvendinimas XILINX Spartan-6

Programos pradžioje įkeliamas AES kodavimo raktas. Rakto duomenims įrašyti į AES kodavimo ir dekodavimo modulius sudaromas skaitiklis, kuris skaičiuoja įrašomus baitus. Raktas įrašomas iškart į abu modulius. Pradėjus skaičiuoti duomenis, nustatomas rakto įvesties signalas ir, suskaičiavus iki 16, šis signalas nutraukiamas.

Įrašius AES rakto duomenis, laukiama „MAVLINK“ protokolo pradžios baito. USART fiziniame duomenų protokolui nuskaityti sudaryti keli skaitikliai. Viena iš jų dalina 50 MHz taktinį dažnį į 9600 [14], lygų duomenų perdavimo greičiui. Kitas skaitiklis skaičiuoja USART protokolo bitus, t.y., pradžios bitą, 8 duomenų bitus ir pabaigos bitą. Kai aptinkamas „MAVLINK“ protokolo pradžios baitas, į registrus įrašomi visi protokolo duomenys. Į kodavimo modulį įkeliami tik protokolo duomenys ir kontrolinės sumos. Blokai, kuriems trūksta 16 bitų, užpildomi nuliais. Kadangi LPLM luste nesudaryta duomenų kaupimo eilė, naudojami RTS/CTS srauto valdymo signalai. RTS signalas yra USART modulio išvesties signalas, kuris nurodo, ar modulis gali priimti duomenis. CTS signalas yra įvadas, kuris nurodo, ar galima išsiųsti duomenis. Taigi priėmus neužkoduotus duomenis, jie koduojami ir iškart siunčiami. Kol vyksta siuntimas, RTS signalas nustatomas veikti 1 pozicijoje, reiškiančioje, kad modulis užimtas ir negali priimti duomenų. Dekodavimo modulio CTS signalu nurodoma, ar dekoduoti duomenys gali būti išsiųsti, t.y., jei signalo reikšmė 1, laukiama, kol jis pasikeis į 0. UART duomenų baitų paruošimo fragmentas VHDL kodoavimo kalba :

```
UART_CLK_DIV_pre : process(CLK)
begin
```

```

if(CLK'event and CLK = '1') then
    if(UART_EN = '1') then
        if(UART_int < UART_cnt_cmp) then
            UART_int <= UART_int + 1;
        else
            UART_START_BIT <= '0';
            UART_int <= 0;
            UART_CLK <= not UART_CLK;
        end if;
    else
        UART_START_BIT <= '1';
        UART_int <= 0;
        UART_CLK <= '0';
    end if;
end if;
end process;

UART_bit_prc : process(UART_CLK)
begin
    if(UART_EN = '0') then
        UART_RESET <= '0';
    elsif(UART_CLK'event and UART_CLK = '1') then
        if(UART_bit > 0 and UART_bit < 9) then
            UART_DAT_TMP(UART_bit-1) <= DATA_IN;
        end if;
        if(UART_bit >= 9) then
            UART_RESET <= '1';
            UART_DATA(UART_byte) <= UART_DAT_TMP;
            if(UART_byte < 15) then
                UART_byte <= UART_byte + 1;
                UART_RECEIVED <= '0';
            else
                UART_byte <= 0;
                UART_RECEIVED <= '1';
            end if;
        end if;
    end if;
end if;

    if(UART_EN = '0') then
        UART_bit <= 0;
    elsif(UART_CLK'event and UART_CLK = '0') then
        UART_bit <= UART_bit + 1;
    end if;
end process;

```

AES kodavimo įgyvendinimas Raspberry pi

Įgyvendinus užduoti LPLM įrenginyje, surandamas būdas realizuoti AES kodavimą „Raspberry pi“ kompiuteryje. Kadangi „Raspbian“ operacinėje sistemoje galima kurti programas įvairiomis programavimo kalbomis, pasirenkama „Python“ kalba, turinti biblioteką, skirtą įvairių tipų kriptografijai – „pycrypto 2.6.1“. Šioje bibliotekoje yra aprašytas ir AES kodavimui pritaikytas algoritmas, kas žymiai palengvina kodavimo algoritmo įgyvendinimą (25 pav.).

```
1 def AES_encode(key,message): #key, message - 16 B
2     obj = AES.new(key, AES.MODE_ECB, 'This is an IV456') #objektas kodavimui
3     encrypted_text = obj.encrypt(message) #uzkoduojama žinutė
4     return encrypted_text #grąžinama reikšmė
5
6 def AES_decode(key,message): #key, message - 16 B
7     obj2 = AES.new(key, AES.MODE_ECB, 'This is an IV456') #objektas dekodavimui
8     decrypted_text = obj2.decrypt(message) #dekoduojama žinutė
9     return decrypted_text #grąžinama reikšmė
10
```

25 pav. AES kodavimo paprogramės python kalba

Pagrindinį protokolo dydį nulemia persiunčiami duomenys (payload). Jie gali kisti nuo 0 iki 255 baitų, todėl užkoduoti AES algoritmu reikia tik „PAYLOAD“ ir „CKA“, „CKB“ sritis. Koduojama 16 baitų grupėmis; jeigu duomenų trūksta, baitai užpildomi 0 vertėmis. UART duomenų nuskaitymas vykdomas tokia tvarka:

```
1 s = sender.read(1) # tikrinamas srautas po vieną baitą
2 if ord(s) == 254: #Tikrinamas pradžios baitas (STX)
3
4     payload_length = ord(sender.read(1)) #LEN
5     packet_sequence = ord(sender.read(1)) #SEQ
6     system_ID = ord(sender.read(1)) #SYS
7     component_ID = ord(sender.read(1)) #COMP
8     message_ID = ord(sender.read(1)) #MSG
9     payload = sender.read((payload_length)+2) #PAYLOAD + CKA + CKB
10
```

26 pav. MAVLink protokolo duomenų nuskaitymas python kalba

```

1 encode_counter = (((payload_length)+2)/16+1)*16 #apskaičiuojamas kodavimo dydis baitais
2 if encode_counter > 16: # ilgio tikrinimas
3     #sukarpomi gauti duomenys po 16 B (užildomi iki 16 B)
4     payload_array = split_string(payload,16)
5
6     for number in range(0,len(payload_array)):
7         #AES uzkodavimas po 16 B
8         payload_array[number] = AES_encode(payload_array[number])
9         #konvertavimas paruošiant išsiųsti baitus
10        full_payload.extend([ord(elem) for elem in payload_array[number]])
11 else:
12     payload = payload.ljust(16,"\\x00") #užpildymas iki 16 B nuliais
13     payload_array = [AES_encode(payload)] #konvertavimas į masyvą
14     #konvertavimas paruošiant išsiųsti baitus
15     full_payload = [ord(elem) for elem in payload_array[0]]
16
17 encoded_answer = msg_parameters + full_payload #paruošiami surinkti duomenys išsiuntimui
18 reciever.write(str(bytearray(encoded_answer))) #išsiunčiamas baitų masyvas per UART
19

```

27 pav. MAVLink duomenų paruošimas AES kodavimui

Apskaičiuojama, kokio ilgio turi būti kodas. Kodavimo skaitiklis išduoda 16, 32, 48 ir t.t. skaičių sekas. Palyginama, ar gautas skaičius didesnis nei 16 baitų. Jeigu didesnis, jis yra sukarpomas po 16 baitų ir užkoduojamas blokais. Jeigu jis mažesnis arba lygus 16 baitų, turinys užpildomas iki 16 baitų nuliais ir užkoduojamas. Įdėjimas į masyvą skirtas tik duomenų tipų suvienodinimui. Tolimesni veiksmai – duomenų konvertavimas ir sudėjimas į masyvą, baitų paruošimas išsiųsti.

```

Real rc: [254, 9, 82, 255, 190, 0, 0, 0, 0, 0, 6, 8, 0, 0, 3, 68, 42]
Enc snt: [254, 9, 82, 255, 190, 0, 67, 133, 68, 167, 100, 186, 25, 119, 221, 157, 50, 251, 110, 25, 50, 227]
Enc rec: [254, 9, 82, 255, 190, 0, 67, 133, 68, 167, 100, 186, 25, 119, 221, 157, 50, 251, 110, 25, 50, 227]
Real de: [254, 9, 82, 255, 190, 0, 0, 0, 0, 0, 6, 8, 0, 0, 3, 68, 42]

```

28 pav. MAVLink duomenų baitų atvaizdavimas

Baitų sraute matome duomenų baitų vertes (28 pav.). „Real rc“ nurodo gautą neužkodotą baitų verčių masyvą, „Enc snt“ – išsiųstą užkodotą baitų masyvą, „Enc rec“ – gautas užkodotas baitų masyvas, „Real dec“ – dekodotas baitų masyvas. Matyti, kad pirmos ir paskutinės eilutės duomenys vienodi. Taip nustatoma, kad sistema veikia ir sėkmingai užkoduoja ir dekoduoja gautus srauto duomenis.

Pirmi 6 baitai išsaugomi ir paliekami nekoduoti. Likę baitai užkoduojami. Šiuo atveju, likusi paketo dalis nėra 16 baitų (sužinoma iš antro baito vertės; jis nurodo duomenų ilgį), todėl ji iki 16 baitų užpildoma nulinėmis vertėmis ir tuomet užkoduojama.

2.3 AES kodavimo įgyvendinimas STM32F103

AES kodavimas įgyvendintas „ARM“ architektūros „STM32F103“ mikrovaldiklyje, kurio visų registrų dydis yra 32 bitai [10]. Šio mikrovaldiklio procesoriaus dažnis nustatytas ties 72 MHz. Mūsų sistemoje naudojamas „MAVLINK“ duomenų perdavimo protokolas. Jis sudarytas iš pradžios baito „FE“, duomenų paketo ilgio, paketų skaitiklio, skirto jų sekimui, 3 identifikacinių baitų, duomenų ir dviejų kontrolinių sumų baitų. Kodavimui duomenis pasirinkime tik „MAVLINK“ duomenų srityje ir kontrolines sumas. Pagrindiniame cikle kreipiamasi į

dvi procedūras: viena iš jų tikrina, ar priimti „MAVLINK“ protokolo paketai kodavimui, kita – ar priimti „MAVLINK“ protokolo paketai dekodavimui.

Mikrovaldiklyje nustatomos įvesties/išvesties jungtys, konfigūruojami „USART“ registrai. Kodavimui naudojamas pirmasis mikrovaldiklio „USART“ blokas. Nustatomas 9600 bps greitis. Naudojamos „A“ prievado jungtys, kurios nustatomos į alternatyvią funkciją, t.y. „USART“. Taip pat konfigūruojamas pertraukčių modulis, kuris generuos pertrauktį kiekvieną kartą gavęs duomenų baitą. Lygiai taip pat sukonfigūruojamas antras USART modulis duomenų dekodavimui.

Pertraukčių procedūroje duomenys suskirstomi pagal „MAVLINK“ protokolą. Procedūra laukia „MAVLINK“ protokolo pradžios baito, kuris lygus „FE“ (šešioliktainis skaičius). Po šio pradžios baito nuskaitomas paketo dydis, paketo eilės numeris, trys identifikaciniai baitai, duomenys ir kontrolinės sumos. Šie protokolo duomenys įrašomi į vieną iš 32 pagal „MAVLINK“ protokolą suskirstytų duomenų struktūrų. Lygiai taip pat pertraukčių procedūra vykdoma ir dekodavimui.

Pagrindiniame cikle kodavimo procedūra laukia, kol bus priimti duomenys, tuomet juos užkoduoja ir išsiunčia. Kai tik nuskaitomas „MAVLINK“ protokolo paketas, jis įrašomas į apdorojimo eilę, galinčią saugoti iki 32 „MAVLINK“ paketų. Priėmus duomenis, sudaroma duomenų išsiuntimo baitų eilė. Pirmieji 6 informaciniai baitai sudedami tiesiogiai į eilę, o „MAVLINK“ protokolo duomenų sritis su kontrolinėmis sumomis užkoduojama AES kodavimu. Kadangi AES duomenis segmentuoja kas 16 baitų, kodatoriaus įvestyje siunčiami „MAVLINK“ protokolo duomenys su kontroline suma ir, dar trūkstant duomenų iki 16 kartotinio skaičiaus, užpildoma nulinais baitais. Taip išėjime gauname užkoduotus duomenis, kurių dydis yra 16 kartotinis skaičius. Šie užkoduoti baitai sudedami į išsiuntimo duomenų eilę ir kai tik procesorius laisvas, išsiunčiami.

Antroji pagrindinio ciklo procedūra priima užkoduotus duomenis ir dekoduoja juos. Priėmus modifikuoto „MAVLINK“ protokolo duomenis, jie taip pat įrašomi į 32 protokolo dydžių eilę. Protokolas modifikuotu vadinamas todėl, kad nurodytas duomenų dydis nėra tikras, o duomenys segmentuoti po 16 baitų. Iš priėmimo eilės pirmi 6 baitai perduodami tiesiogiai į išsiuntimo duomenų eilę, o užkoduoti duomenys yra dekoduojami. Kadangi dekodavus dažnu atveju gaunamas duomenų perteklius, į išsiuntimo eilę paduodami ne visi duomenys, o tik pagal „MAVLINK“ protokolo antru baitu nurodytą paketo dydį ir papildomai 2 kontrolinės sumos. Taip išlaikomas „MAVLINK“ protokolo standartas, tačiau padidėja duomenų srautas. Duomenų srauto suvaldymui sudaryta 32 protokolo dydžio eilė, kurioje kaupiami duomenys, jei dar ankstesnių duomenys nebuvo užkoduoti/dekoduoti ir išsiųsti.

2.4 Dirbtinių neuronų tinklo įgyvendinimas

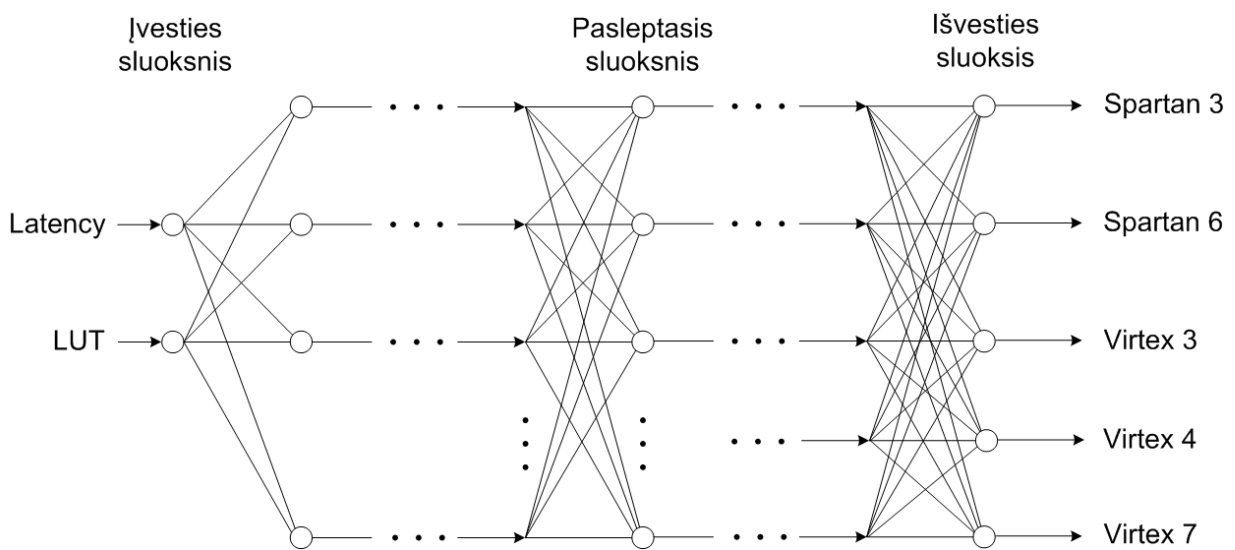
Kai jau galima užkoduoti reikiamus duomenis, sukuriamas neuroninis tinklas, kuris pagal parašytos VHDL programos kodo užimtumą LPLM procesoriuje praneša, koks LPLM modelis kiekvienu atveju labiausiai tinkamas. Kadangi su vienu LPLM lygiagrečiai galima atlikti kelis veiksmus, koduojami keleto bepiločių orlaivių duomenų srautai. Priklausomai nuo to siūlyomas

optimalus LPLM modelis. Tokios sistemos sukūrimui reikės dviejų įvesčių, tai yra LUT ir „Latency“.

- LUT – užimtų loginių blokų skaičius LPLM įrenginije;
- „Latency“ – laikas, per kurį atliekamas veiksmas, skaičiuojama pagal formulę:

$$\alpha = \frac{t}{if_{max}}$$

kai i – impulso trukmė, t – kodavimo trukmė, f_{max} – maksimalus dažnis



29 pav. Sukurtas daugiasluoksnis perceptronas

Simuliuojant VHDL kodą, sužinomas teorinį LPLM veikimo dažnį ir kokia LUT apimtis naudojama. Paskaičiuojamas „latency“ ir šie duomenys įkeliami į parašytą „Matlab“ programą. Su šia informacija apmokome daugiasluoksnį parceptroną, turintį 8 paslėptus neuronų sluoksnius, tam skyrėme 100 iteracijų. Apmokę neuroninį tinklą (29 pav.), įkeliamo duomenis sistemos patikrinimui.

```
clear all; close all; clc;
```

```
P = [...
```

```
[2.26*1 2.26*2 2.26*3 2.26*4 2.26*5 1.41*1 1.41*2 1.41*3 ...
1.41*4 1.41*5 1.01*1 1.01*2 1.01*3 1.01*4 1.01*5 0.84*1 ...
0.84*2 0.84*3 0.84*4 0.84*5 0.78*1 0.78*2 0.78*3 0.78*4 ...
0.78*5])/11;% Latency [us] x itaisu sk.
```

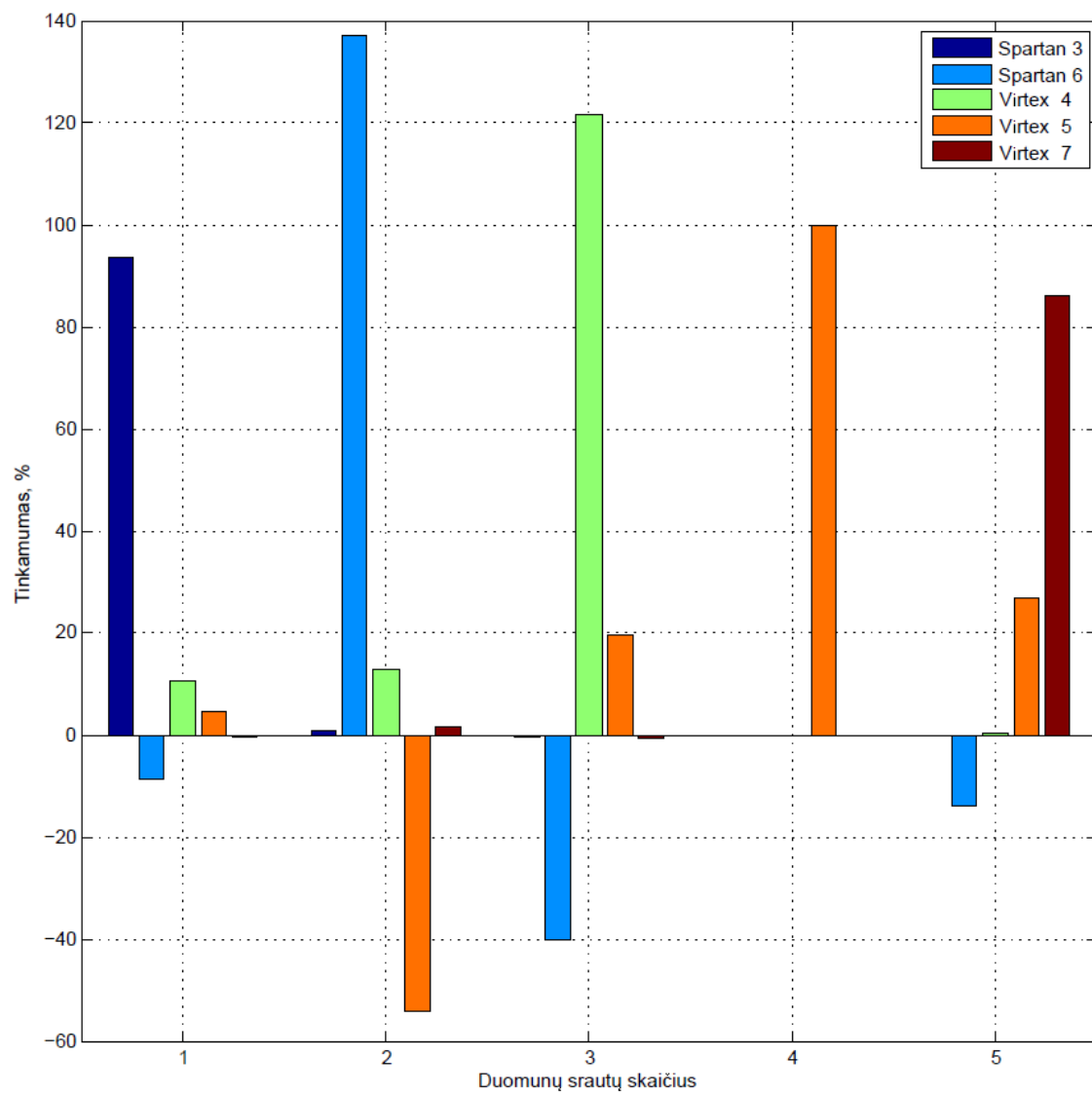
```
[0.41 0.41 0.41 0.41 0.41 0.1 0.1 0.1 0.1 0.1 0.16 0.16 0.16 ...
0.16 0.16 0.08 0.08 0.08 0.08 0.08 0.01 0.01 0.01 0.01 ...
0.01]/0.2]; % LUT [%]
```

```

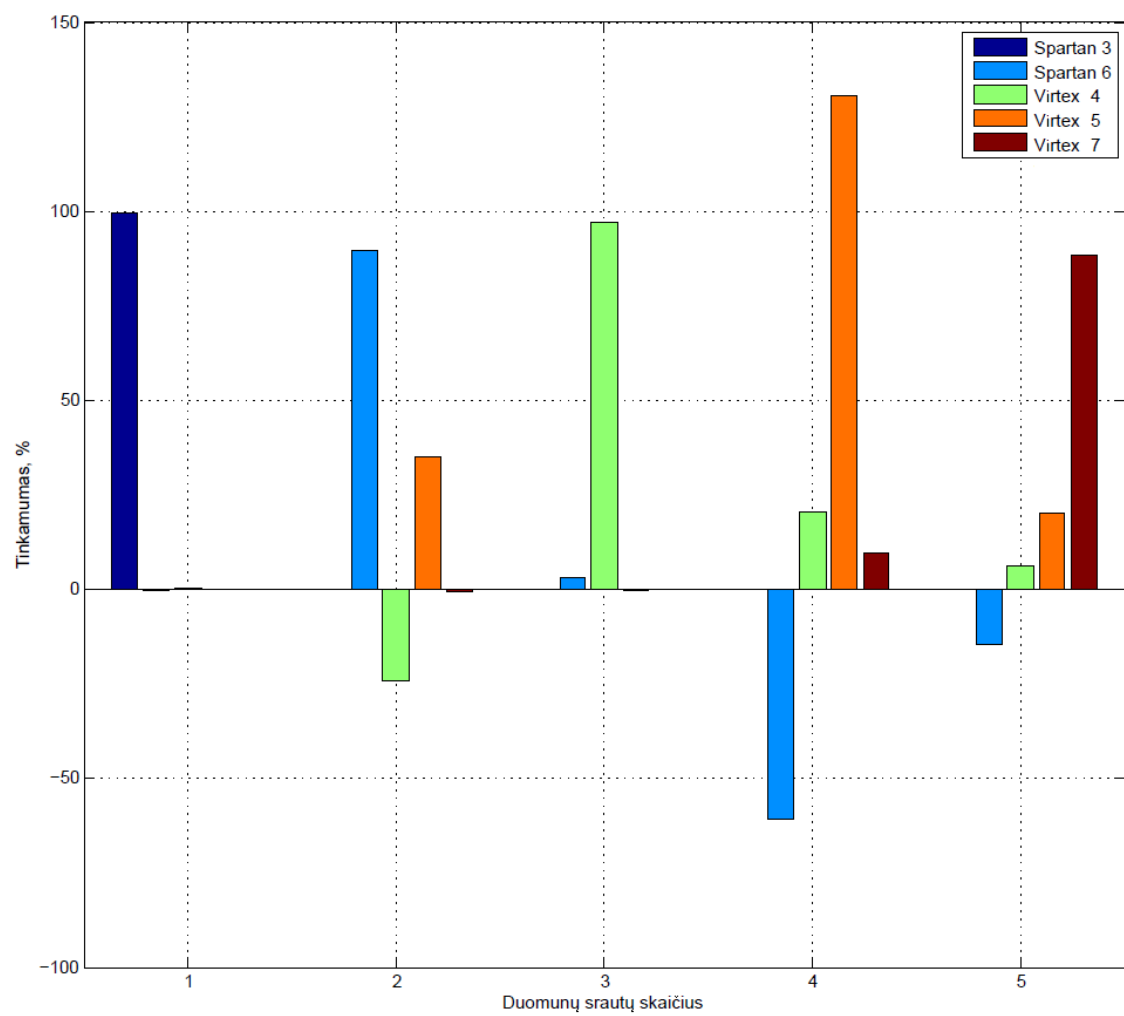
Ts3 = [1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
Ts6 = [0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0];
Tv4 = [0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0];
Tv5 = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0];
Tv7 = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1];
TT = cat(1,Ts3,Ts6);
TT = cat(1,TT,Tv4);
TT = cat(1,TT,Tv5);
TT = cat(1,TT,Tv7);
pp = [...[3 3 3 3]*0.01; % N itaisu
        [2.2*5 1.50*1 0.96*2 0.84*3 0.75*4]/11;
        %[v4 s6 s3] % testavimo duomenys, Latency [us] x itaisu sk.
        [0.4 0.12 0.17 0.08 0.02]/0.2];
        %[v4 s6 s3] % testavimo duomenys, LUT [%]
net = feedforwardnet(8);
net = configure(net,P,TT);
net.trainParam.epochs = 100;
[net,Y,E] = train(net,P,TT);
y = sim(net,pp);
figure(3);
stem(y,'r');
bar(100*[y'],'grouped'); grid on; ylabel('Tinkamumas,%'); ...
xlabel('FPGA_tipas');

```

Paleidus „Matlab“ programą gauname grafiką (30, 31 pav.), kuriame daugiasluoksnis perceptronas pasiūlo tinkamiausią LPLM modulį priklausomai nuo laiko, reikalingo vienam blokui užkoduoti, ir užimtų loginių blokų skaičiaus LPLM įrenginyje. Kuriant naują sistemą, sudarytą iš LPLM, galima nesunkiai patikrinti, kuris iš LPLM įrenginių yra labiausiai tinkamas.



30 pav. Neuroninio tinklo pasirinktas LPLM įtaisas pagal duomenų kodavimo srauto kiekį



31 pav. Neuroninio tinklo pasirinktas LPLM įtaisas pagal duomenų dekodavimo srauto kiekį

3 DUOMENŲ ŠIFRAVIMO ALGORITMO TYRIMAS

3 ir 4 lentelėse pateiktas penkių LPLM mikrograndynų vieno bloko kodavimo ir dekodavimo trukmės laikas. Šis laikas priklauso nuo maksimalaus mikrograndyno pasiekiamo dažnio ir kodavimui/dekodavimui reikalingo impulsų skaičiaus. Taip pat simuliuojant kodavimo algoritmą sužinoma, kiek loginių elementų turi tiriamas LPLM mikrograndynas ir kiek jų naudojama kodavimui įgyvendinti. Norint suskaičiuoti, kiek laiko užtrunka užkoduoti vieną duomenų bloką, kurį sudaro 16 baitų, naudojama tokia formulė:

$$v_{AES} = \frac{16}{\alpha}.$$

Lentelė 3. Kodavimo algoritmo įgyvendinimo parametrai

Encryption AES-128					
LPLM	Spartan-3	Spartan-6	Virtex-4	Virtex-5	Virtex-7
Latency(us)	2,27	1,41	1,01	0,84	0,78
f _{max} (MHz)	123	198	277	332	357
Taktinių impulsų skaičius (clock)	279	279	279	279	279
LUT's (vnt)	1591	943	1801	1095	940
Available LUT's (vnt)	3840	9112	10944	12480	204000
Užimtumas (%)	41	10	16	8	1
1 AES bloko sparta (MB/s)	7,05	11,35	15,89	19,04	20,47
AES skaičius viename FPGA	2	10	6	12	100
Maximali fpga sparta (MB/s)	14,11	113,55	95,31	228,47	2047,31

Lentelė 4. Dekodavimo algoritmo įgyvendinimo parametrai

Decryption AES-128					
LPLM	Spartan-3	Spartan-6	Virtex-4	Virtex-5	Virtex-7
Latency(us)	2,47	1,95	1,55	1,00	1,11
f _{max} (MHz)	113	143	180	278	251
Taktinių impulsų skaičius (clock)	279	279	279	279	279
LUT's (vnt)	1793	1302	1869	1259	1369
Available LUT's (vnt)	3840	9112	10944	12480	204000
Užimtumas (%)	46	14	17	10	1
1 AES bloko sparta (MB/s)	6,48	8,20	10,32	15,94	14,39
AES skaičius viename FPGA	2	7	5	10	100
Maximali fpga sparta (MB/s)	12,96	57,41	51,61	159,43	1439,43

Taip pat lentelėse nurodyti algoritmo implementavimui panaudoti kiekvieno LPLM įtaiso loginiai elementai (jų skaičius). Žinant vieno įtaiso bendrą loginių įtaisų skaičių, galima paskaičiuoti, kiek kodavimo algoritmų galima įgyvendinti lygiagrečiai. Pastebėta, kad „Virtex-4“ LPLM įtaisas, nors ir daug pranašesnis už „Spartan-6“, algoritmo įgyvendinimui naudoja daug daugiau loginių elementų. Tai yra dėl to, kad „Virtex-4“, kaip ir „Spartan-3“, naudoja kitokią mikrograndyno architektūrą. Paskaičiavus vieno LPLM įtaiso algoritmo užimtumo procentą galima sužinoti, kiek šifravimo algoritmų vienu metu gali vykdyti vienas LPLM įtaisas. Turint šiuos duomenis išvedama teorinė vieno LPLM šifravimo sparta.

3.1 Vaizdo duomenų kodavimas

Koduojant vaizdo duomenis susiduriama su kita problema: priklausomai nuo rezoliucijos ir kadrų per sekundę (angl. frames per second) perduodami skirtingi duomenų kiekiai, todėl norint parinkti tinkama kodavimo įrenginį reikia paskaičiuoti, kokia didžiausia kodavimo sparta įtaisai galės koduoti duomenis, neužlaikydami vaizdo transliavimo. Tai galima paskaičiuoti pagal lygtį:

$$v_{LPLM} = \frac{LUT_{max} \cdot v_{AES}}{LUT},$$

kai LUT – kodavimo algoritmo loginių elementų skaičius, LUT_{max} – loginių elementų skaičius pasirinktoje LPLM. Kadangi LPLM lustas gali vykdyti programą ne tik nuosekliai, bet ir lygiagrečiai, likę neišnaudoti loginiai elementai gali atlikti tą pačią funkciją vienu metu. Todėl programos vykdymo laikas sutrumpėja tiek kartų, kiek LPLM gali vykdyti programų vienu metu (tai priklauso nuo loginių elementų skaičiaus). Žinant šiuos duomenis galima paskaičiuoti, kiek neglaudinto video duomenų kanalų galima koduoti su viena iš šių „LPLM“ mikrograndinių. Skaičiuojama, kokia sparta perduodami neglaudinto video duomenys, priklausomai nuo rezoliucijos ir kadrų per sekundę skaičiaus:

$$v_{CH} = r \cdot fps,$$

kai r – pikseliai, fps – kadrai per sekundę. Pagal kitą formulę paskaičiuojama, kiek kanalų gali koduoti vienas LPLM mikrograndynas, lygiagretinant programos veikimą:

$$N_{CH} = \frac{v_{LPLM}}{v_{CH}}.$$

Lentelė 5. Vaizdo duomenų kodavimo spartos

				Spartan-3	Spartan-6	Virtex-4	Virtex-5	Virtex-7	STM	Raspberry Pi
Maximali sparta				14,11	113,55	95,31	228,47	2047,31	3,17	52,98
Resolution	fps	Greitis (B/s)	Greitis (MB/s)							
480*360	10	1728000	1,65	8	68	57	138	1242	1	32
	20	3456000	3,30	4	34	28	69	621	0	16
	30	5184000	4,94	2	22	19	46	414	0	10
640*480	10	3072000	2,93	4	38	32	77	698	1	18
	20	6144000	5,86	2	19	16	38	349	0	9
	30	9216000	8,79	1	12	10	25	232	0	6
1280*720	10	9216000	8,79	1	12	10	25	232	0	6
	20	18432000	17,58	0	6	5	12	116	0	3
	30	27648000	26,37	0	4	3	8	77	0	2
1920*1080	10	20736000	19,78	0	5	4	11	103	0	2
	20	41472000	39,55	0	2	2	5	51	0	1
	30	62208000	59,33	0	1	1	3	34	0	0

Lentelė 6. Vaizdo duomenų dekodavimo spartos

				Spartan 3	Spartan6	Virtex 4	Virtex 5	Virtex 7	STM	Raspberry pi
Maximali sparta				12,96	57,41	51,61	159,43	1439,43	6,44	107,38
Resoliution	fps	Greitis (B/s)	Greitis (MB/s)							
480*360	10	1728000	1,65	7	34	31	96	873	3	65
	20	3456000	3,30	3	17	15	48	436	1	32
	30	5184000	4,94	2	11	10	32	291	1	21
640*480	10	3072000	2,93	4	19	17	54	491	2	36
	20	6144000	5,86	2	9	8	27	245	1	18
	30	9216000	8,79	1	6	5	18	163	0	12
1280*720	10	9216000	8,79	1	6	5	18	163	0	12
	20	18432000	17,58	0	3	2	9	81	0	6
	30	27648000	26,37	0	2	1	6	54	0	4
1920*1080	10	20736000	19,78	0	2	2	8	72	0	5
	20	41472000	39,55	0	1	1	4	36	0	2
	30	62208000	59,33	0	0	0	2	24	0	1

Remiantis turimais rezultatais galima teigti, kad kodavimo algoritmas, įgyvendinamas LPLM lustuose, pasiekia didesnę kodavimo spartą nei dekodavimo algoritmas. Tai galima būtų paaiškinti tuo, kad dekodavimo algoritmo įgyvendinimui reikalingas didesnis loginių elementų skaičius, todėl dekodavimo sparta mažėja. Palyginus LPLM įtaisų spartą su „STM“ mikrovaldikliu ir „Raspberry pi“, pastebimas atvirkštinis rezultatas. Kitaip nei LPLM įtaisuose, kodavimo sparta yra kone dvigubai mažesnė nei dekodavimo. Šuos skirtumus lemia algoritmo įgyvendinimas.

4 APIBENDRINIMAS. IŠVADOS

Magistro baigiamajame darbe iškelti tikslai pasiekti ištyrus realiu laiku veikiančių duomenų šifravimo algoritmo įgyvendinimų LPLM įrenginiuose galimybes. Apžvelgus analitinę literatūrą, įgyvendinus duomenų šifravimo algoritmą skirtinguose LPLM įrenginiuose ir ištyrus jų veikimą, pateikiamos šios švados:

1. Atlikus analitinę literatūros apžvalgą daromos šios išvados:

- Išrinktas AES duomenų kodavimo algoritmas, kuris gali būti įgyvendinamas skirtinguose mikrograndynų architektūrose ir yra atsparus grubaus slaptažodžių laužimo atakoms (angl. brute force attack).
- LPLM įrenginiai yra tinkami pasirinktam kodavimo algoritmo įgyvendinimui, nes turi pakankamai paskirstytų loginių resursų, kuriuos naudijant išlygiagretinus kodavimo algoritmą galima šifruoti duomenis realiu laiku.

2. LPLM įrenginyje įgyvendinus duomenų šifravimo algoritmą bei atlikus eksperimentinę jos patikrą, daromos šios išvados:

- LPLM įrenginyje VHDL aparatine programavimo kalba įgyvendintas AES kodavimo/dekodavimo algoritmas, kuris gali veikti realiu laiku ir atlikti skaičiavimus per 279 taktinius impulsus.
- Sukurtas daugiasluoksnis perceptronas pasiūlo tinkamiausią LPLM modulį priklausomai nuo vieno bloko kodavimui reikalingo laiko ir nuo užimtų loginių blokų skaičiaus LPLM įrenginyje.
- Visi tirti šifravimo algoritmai, įgyvendinti LPLM įtaisuose, pranoksta algoritmo „STM“ mikrovaldiklyje įgyvendinimą pagal duomenų šifravimo spartą.
- Kodavimo algoritmas, veikiantis „Virtex-5“ LPLM įtaise, pagal greitaveiką 5 kartus pranoksta algoritmą, veikiantį „Raspberry Pi“ mikrokompiuteryje.
- Norint koduoti neglaudintą 1080×1920 raiškos ir 30 kadrų per sekundę vaizdo srautą galima taikyti „Spartan-6“ ir „Virtex“ šeimos LPLM.
- Norint dekoduoti neglaudintą 1080×1920 raiškos ir 30 kadrų per sekundę vaizdo srautą galima taikyti „Virtex-5“, „Virtex-7“ LPLM arba „Raspberry Pi“ mikrokompiuterį.
- Telemetrinių duomenų maksimaliai spartai esant iki 3,17 MB/s, signalo kodavimui galima taikyti STM mikrovaldiklį.
- „Spartan-3“ LPLM leidžia koduoti vieno kanalo vaizdo srautą, kurio raiška 1280×720 ir dažnis 10 kadrų per sekundę.

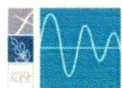
Literatūros sąrašas

- [1] J. Daemen and V. Rijmen, *The Design of Rijndael: AES—the Advanced Encryption Standard*. Berlin ; New York: Springer, 2002.
- [2] A. Soltani and S. Sharifian, “An ultra-high throughput and fully pipelined implementation of AES algorithm on FPGA,” *Microprocessors and Microsystems*, vol. 39, pp. 480–493, Oct. 2015.
- [3] C.-h. Liu, J.-s. Ji, and Z.-l. Liu, “Implementation of DES Encryption Arithmetic based on FPGA,” *AASRI Procedia*, vol. 5, pp. 209–213, 2013.
- [4] I. Kuon, R. Tessier, and J. Rose, *FPGA Architecture: Survey and Challenges*, vol. 2. 2007.
- [5] V. Abraitis, E. Bareiša, and R. Benisevičiūtė, “The Testing Methods of Programmable Integrated Circuits,” in *Elektronika Ir Elektrotechnika*, vol. 47, 2015.
- [6] “Verilog Resources.” <http://www.verilog.com/> [žiūrėta 2016-05-10].
- [7] D. L. Perry, *VHDL: Programming by Example*, vol. 4. McGraw-Hill, 2002.
- [8] “Verilog’s inventor nabs EDA’s Kaufman award.” http://www.eetimes.com/document.asp?doc_id=1157349 [žiūrėta 2016-05-10].
- [9] E. Upton and G. Halfacree, *Raspberry Pi User Guide*. John Wiley & Sons, Sept. 2014.
- [10] S. STM32F101xx and S. STM32F103xx, “And STM32F107xx advanced ARM-based 32-bit MCUs,” *STMicroelectronics, RM0008*, 2015.
- [11] “MAVLink Micro Air Vehicle Communication Protocol - QGroundControl GCS.” <http://qgroundcontrol.org/mavlink/start> [žiūrėta 2016-12-13].
- [12] “What are Artificial Neural Networks.” <http://www.psych.utoronto.ca/users/reingold/courses/ai/cache/neural2.html>. [žiūrėta 2016-05-10].
- [13] E. J. Miles, “SHALLOW WATER QUALITY MONITORING.” http://www.vims.edu/cbnerr/_docs/monitoring_docs/chpt6.pdf, 2009. [žiūrėta 2016-05-10].
- [14] M. A. Mohamed, “FPGA synthesis of VHDL OFDM system,” *Wireless personal communications*, vol. 70, no. 4, pp. 1885–1909, 2013.

PRIEDAI

A priedas. Pranešimo 20-oje Lietuvos jaunųjų mokslininkų konferencijoje pažyma

B priedas. Straipsnio „Efficient Path Planning Methods for UAVs Inspecting Power Lines“
IEEE medžiaga



VILNIAUS GEDIMINO
TECHNIKOS UNIVERSITETAS
ELEKTRONIKOS FAKULTETAS

XX-toji Lietuvos jaunųjų mokslininkų konferencija
„MOKSLAS – LIETUVOS ATEITIS“
ELEKTRONIKA IR ELEKTROTECHNIKA

PAŽYMA

Vladas Taluntis

*Sistemų inžinerijos ir kompiuterių technologijų (T120),
Signalų technologijų (T121), Aukštų dažnių technologijų
(T191) jungtinėje sekcijoje perskaitė pranešimą*

TELEMETRINIŲ DUOMENŲ ŠIFRAVIMO ALGORITMŲ ĮGYVENDINIMO LPLM IRENGINIŲ TYRIMAS

Mokslo komiteto pirmininkas

prof. habil. dr. Romanas Martavičius

Sekcijos pirmininkas

dr. Darius Plonis

2017 m. kovo 17 d.
Vilnius

Efficient Path Planning Methods for UAVs Inspecting Power Lines

¹Vytautas Rafanavicius, ¹Piotras Cimmerman, ¹Vladas Taluntis, ²Ka Lok Man, ¹Gintaras Volkvicius, ¹Martynas Jurkynas, ¹Justas Bezarus

¹Baltic Institute of Advanced Technology, Lithuania, vytautas.rafanavicius@bpti.lt, piotras@bpti.lt

²Xi'an Jiaotong-Liverpool University, China

Abstract - Power line inspection is one of the most difficult and time consuming steps in power line maintenance. Even for a sizeable group of workers it takes months to inspect all of them, especially when they are not visible from the road and must be inspected on foot or with an aerial vehicle. That problem is even more prominent when the inspection must be done as fast as possible when the power cuts out in certain regions after natural disaster. To save time and reduce expenditure Unmanned Aerial Vehicles (UAV) could be used to film the power lines and automatically find problems (e.g. a broken cable or a tree branch too close to the a line). Our research focuses on planning the route for the survey.

Keywords: UAV, power line, path planning, graph

1. Introduction

The power lines inspection by its importance is divided into two:

Periodic. A team is sent to inspect electricity lines and to look for defects and objects too close to the lines [1],[2],[3]. They report problems to headquarters where they plan further actions and send a team to fix the issues.

Critical. In a state of emergency, when a line is broken after a storm or other events, a helicopter is sent to find the faulty power line (if the exact position of the fault is not reported by the local people). They report problems to the headquarters where they make a decision to send teams to reported locations in order to fix the issues.

Before any UAV is could be used, a path with GPS coordinates must be loaded in its board computer. However, the structure of the power line is not just a line, in most cases it has a number of branches. If UAV relies on fuel or batteries it is a demand to have the most efficient path to cover as many kilometres of the power line as possible. To plan a path the structure of electricity poles and lines have to be simplified to one that could be easily processed by computer to do any calculations needed. The easiest way to simplify the structure is to make it a graph. In mathematics, and more specifically in graph theory, a graph is a representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by mathematical abstractions called vertices (also called nodes or points), and the links that connect some pairs of vertices are called edges (also called arcs or lines). Electricity poles could be vertices and the lines that

connect them could be edges. Planning a path for UAV is easier than for any ground vehicle, because there is much lower possibly of obstructions in air space than on ground, so there is no need to take account on any obstructions.

Depth-first search algorithm. Depth-first search is an algorithm for traversing or searching tree or graph data structures. One starts at the root (searching some arbitrary vertex as the root in the case of a graph) and explores as far as possible along each branch before backtracking. In this case, search would have to go through all graph's vertices (electricity poles). First it is necessary to mark all vertices from 1 to n (n – is the number of vertices in the graph). Then start trip from the vertex number one, mark it as visited and continue trip to the vertex of last visited vertex's environment, which has the lowest number and is not visited. If trip ends up in vertex that all his environment vertices are visited, search goes back until it finds not visited one. In the end, search ends up at the first vertex [4]. The pseudo code of the recursive type of this algorithm:

procedure DFS-iterative(G,v):

 let S be a stack

 S.push(v)

 while S is not empty

 v = S.pop()

 if v is not labeled as discovered:

 label v as discovered

 for all edges from v to w in

G.adjacentEdges(v) do

 S.push(w)

Shortest Path Search in Unweighted Graph. This algorithm finds the shortest path from the start vertex to other vertices. It uses Breadth-First search (BFS) algorithm. BFS is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary vertex of a graph) and explores the neighbour nodes first, before moving to the next level neighbours [5]. In overall, this algorithm is great and fast solution for finding shortest paths from one point to another, which includes all points and lines in its way. In addition, this algorithm does not consider distances between vertices, so it is simpler and less time consuming than the other "shortest paths" algorithms that will be described next.

Dijkstra's algorithm. It is an algorithm for finding the shortest paths between vertices in a graph, which may represent, for example, road networks or electricity pole networks. This algorithm in principal is very similar to the previous one, it's just that weights of edges are put in the distance array. Also algorithm replaces existing value in array distance array if it calculated the shorter path by going the another way [6]. In this algorithm, closer vertices has higher priority. In addition, instead of queue that is used in BFS algorithm, this algorithm use another array that contains information if it already have visited vertex or not. The search ends after it visits all vertices [7].

This algorithm is perfect if distances between the points are different. In case all distances between certain points are known, this algorithm can find the shortest paths. If the distance between the points are the same or varies slightly, it is better to use Shortest path search in unweighted graph algorithm for finding shortest paths, because it is less time consuming and has better performance.

Fleury's Algorithm. In graph theory, an Eulerian path is a trail in a graph which visits every edge exactly once. Similarly, an Eulerian circuit or Eulerian cycle is an Eulerian trail that starts and ends on the same vertex. Leonhard Euler first discussed it while solving the famous Seven Bridges of Königsberg problem in 1736. The main condition for the graph to have either an Eulerian path or Eulerian cycle is that either every single of its vertices have the even degree (a number of edges touching the vertex) or just two of them have odd degree.

The algorithm for finding either an Eulerian path or Eulerian cycle for the graph that meets the condition is the Fleury's algorithm. With this algorithm, consider a graph known to have all edges in the same component and at most two vertices of odd degree. The algorithm starts at a vertex of odd degree, or, if the graph has none, it starts with an arbitrarily chosen vertex. At each step it chooses the next edge in the path to be one whose deletion would not disconnect the graph, unless there is no such edge, in which case it picks the remaining edge left at the current vertex. It then moves to the other endpoint of that vertex and deletes the chosen edge. At the end of the algorithm, there are no edges left, and the sequence from which the edges were chosen forms an Eulerian cycle if the graph has no vertices of odd degree, or an Eulerian trail if there are exactly two vertices of odd degree [8].

Hierholzer's algorithm specializes in finding an Eulerian cycle in the graph, so the graph has to meet the condition of even vertices degrees to work. The sequence of the algorithm is as follows: any starting vertex v is chosen and a trail of edges from that vertex until returning to v is computed. It is not possible to get stuck at any vertex other than v , because the even degree of all vertices ensures that,

when the trail enters another vertex w there must be an unused edge leaving w . The tour formed in this way is a closed tour, but may not cover all the vertices and edges of the initial graph. As long as there exists a vertex u that belongs to the current tour but that has adjacent edges is not part of the that tour, it has to start another trail from u , following unused edges until returning to u , and join the tour formed in this way to the previous tour [9].

To sum up, this algorithm has far better performance rate for finding an Eulerian cycle.

2. Methodology

Data and Methods for Path Planning. In this project, open source software *Mission Planner* was used, in which route could be planned by simply selecting electrical poles one by one. Exists a few ways to plan the route in *Mission Planner*. One way is to simply select points on the map in order in which they should be visited. The other way is to upload a file with the coordinates of the poles. This file contains lines with points that will be visited. In first line, there is first point with its coordinate, in second line second point and so on. This way is much more efficient.

Reading data of electricity poles. First, some sort of file is needed that would contain each electricity pole's coordinates and information about which poles are adjacent to it. To be efficient, create some sort of data structure to instantly put all poles in while reading them from a file. Two simple data structures could help. First data structure, call it *Position*, will have just two attributes – two doubles for containing coordinates. The second data structure, call it *Pole*, will have three attributes – two doubles for containing coordinates and a vector list of *Position* data's structure objects, that will contain all adjacent poles for the *Pole* object. To even more simplify reading data from file, create a file that would already contains GPS coordinates of the poles and the adjacent ones, so that there would be no need to convert them from any other format.

These two data structures are good for reading data from coordinate file, but not very good for searching for paths.

As there was mentioned in path planning methods section – when searching for the path, a graph with two arrays has to be defined: $L[]$ and $\backslash lst[]$. First, $\backslash L[]$ array will contain adjacent poles from *Pole* structure, and each member of $\backslash lst[]$ array will contain the number of adjacent poles from current *Pole* structure plus the previous array number. For example, there is 5 electricity poles as in Fig. 1, each named from 1 to 5.

The connection between each pole describes this structure of adjacency:

- 1: {2,4}

- 2: {1,5}
- 3: {4, 5}
- 4: {1, 3}
- 5: {2,3}

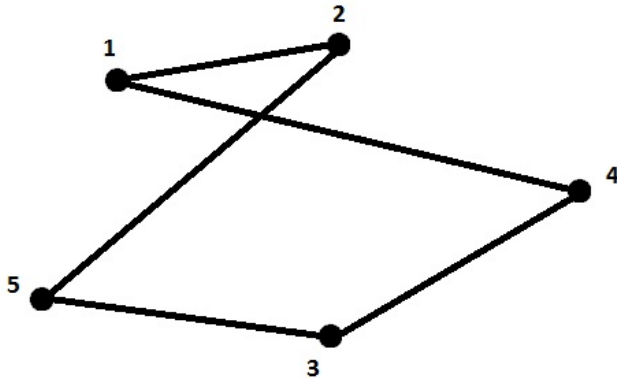


Fig.1 Example of electricity poles graph

Array $L[]$ looks like this: 2 4 1 5 4 5 1 3 2 3

Array $lst[]$ looks like this: 0 2 4 6 8 10

First member of $lst[]$ array describes that there is a graph and this is essential because second number will be calculated as this member plus the adjacent poles for the first pole. Every other number in general is calculated as a previous member plus adjacent poles for the current pole. With this form it becomes like address array: first pole's adjacent poles from array $L[]$ are in an interval $(0;2]$; second pole's adjacent poles from array $L[]$ are in an interval $(2;4]$ and so on.

Formatting an output file. After applying a path planning method an array will be formed, in which the poles that are needed to be visited will be written in visiting order. Now there is a need to print them in lines with any essential information needed for *Mission Planner* to read the output file correctly. Because the visiting order was calculated by the algorithm, program will just search for each member of the path array in *Pole* structure array and when the match is found will print all the information needed in *Mission Planner* syntax.

Overall program creation. An overall program creation could be shown from a programmer point of view with an UML diagram in Fig. 2.

The goal is to visit all poles and all lines, the best algorithm to choose is Hierholzer's algorithm for Eulerian cycle search. Of course, some corrections to main data should be made if it does not meet Hierholzer's algorithm requirements. For this algorithm implementation and overall program writing, choose C++ as program's programming language and *Microsoft Visual Studio 2013* as software for creating it. For test, our own built unmanned aerial vehicle was used, which

is shown in the Fig.3. It has integrated PIXHAWK autopilot system, which controls the plane. This micro-

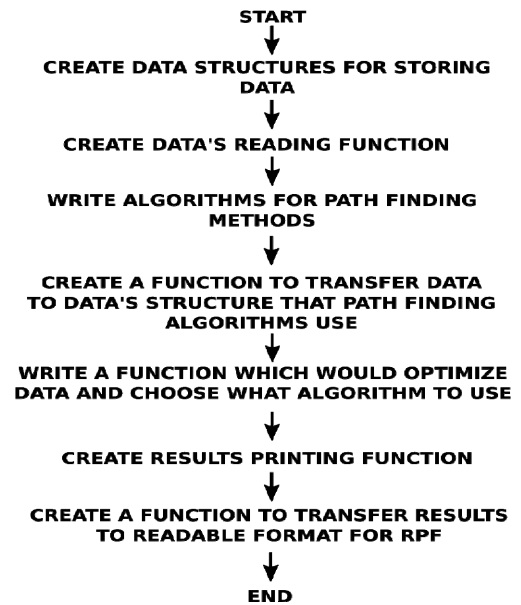


Fig.2 UML diagram of the computer software

controller has a 32-bit Cortex M4 core processor and a set of integrated sensors:

- ST Micro L3GD20H 16 bit gyroscope
- ST Micro LSM303D 14 bit accelerometer / magnetometer
- Invensense MPU 6000 3-axis accelerometer / gyroscope
- MEAS MS5611 barometer



Fig.3 Photo of Unmanned Aerial Vehicle used in this project

UAV was powered by lithium-ion battery. Battery specifications are the following:

- Capacity (mAh): 10000
- Voltage (V): 14.8
- Weight (g): 804

The plane has a gopro hero 3+ camera on board. During the test flight, camera was configured on high quality resolution: 4096 x 2160 and 12 fps video format.

The Flight. Video was filmed from about 70 m height and pointing perpendicularly to the ground. The plane had to visit all electricity poles in its flight path. The UAV was designed for belly landing. The automated landing procedure goes as follows: after the last checkpoint of the mission the descent phase begins. The autopilot turns the plane upwind, throttles down the engine and starts the descend. An important parameter is the glide slope which is the ratio between distance and height from the last checkpoint and desired landing spot. The recommended glide slope depends on the plane model. At a certain height or time to landing threshold the plane flares to increase drag and slow itself before contact with the ground. This method of landing is most useful to us because it is already implemented in Mission Planner, to use it one just needs to specify the landing point and altitude at the end of the route as well as to tweak some settings according to the plane the altitude to flare at, the time to flare before the ground (redundancy is for safety measures) and minimum pitch target.

3. Experiment, Simulation and Analysis

Path planning software. The main goal was to create a program which could quickly plan the shortest and the most efficient route among selected electrical poles.

The program was written using *Microsoft Visual Studio 2013* software in C++ programming language. The program uses Hierholzer's algorithm to find shortest path and data's optimization algorithm to make data valid. This program creates and works with data structures called *Position*, *Pole*, *Structure* and *virtualEdges*. In *Position* the program saves each pole's GPS coordinates on the map (longitude and latitude). In *Pole* it saves each pole's position and information on what other electricity poles that it is connected to. *Structure* contains an array of *Poles* and its status (either it was visited or not). *virtualEdges* is an optimization structure.

Hierholzer's algorithm implementation. Firstly, the data is valid - all vertices degrees are even. Trail starts from first vertex that is in the structure of adjacency and continues in order of the structure of adjacency until it gets back to the beginning (all vertices degrees are even, so that it cannot stuck anywhere else). While travelling vertex by vertex, algorithm deletes edges that just have been travelled through from the structure of adjacency. Also it marks

which vertices it visited and gives priority for those vertices that are not yet visited (if all vertices were visited from given vertex, algorithm travels to the first one of structure of adjacency). When it gets back to the beginning it looks if there are any unvisited vertices. If so, it starts new cycle from it and when finishes it, inserts this tour into the main tour.

With valid data, Hierholzer's algorithm can handle route planning itself. A planned route from valid data in Fig. 4.

While planning this route, algorithm first planned the main big circle and then with second, third and fourth loop planned two triangles and rectangle and inserted them in the main route.

Data optimization algorithm implementation. Implemented Hierholzer's algorithm can plan route when data is valid (all vertices degrees are even). However, in reality could happen that data will not be valid, so there is a demand to write data optimization algorithm, which could make all vertices degrees even from given structure of adjacency by connecting odd degree vertices in pairs so that the sum of new created edges length is shortest. At the start, the algorithm check if there is any odd degree vertices. If so, it recursively, one after another, find all available pairs of vertices and save just the shortest one. To calculate distance between vertices (which are defined in GPS coordinates) algorithm uses haversine formula, which count distance between two GPS points.

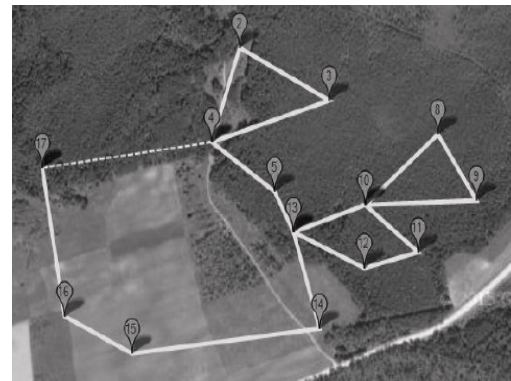


Fig.4 Planned route from valid data

When data is invalid, data's optimization algorithm adds virtual edges to connect odd degree vertices. There is a planned route on electrical poles and lines, which originally formed Y form, in Fig. 5a.

Before planning this route, data's optimization algorithm connected 5th with 4th and 7th and 8th vertices correct data. While first connection can be clearly seen, the second one is on the path in the map that already exists. As one can see in Fig. 5b, when path is moved a little bit, the clear route appears.

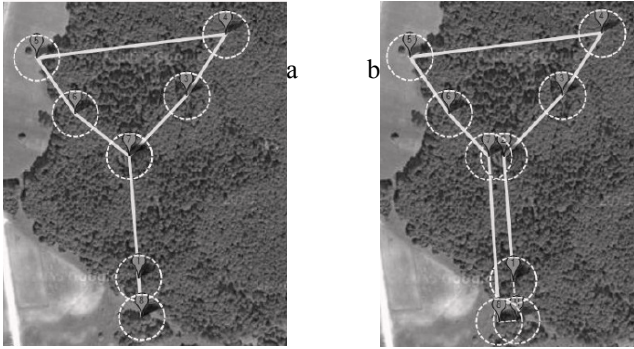
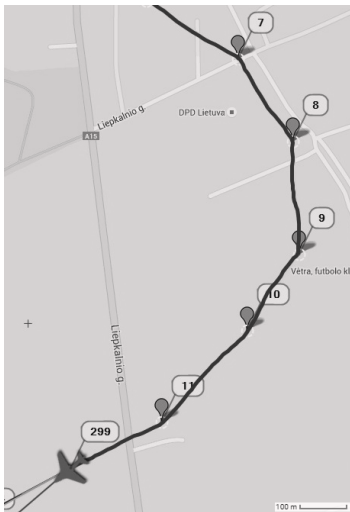


Fig.5 Planned route using not valid data (a), moved planned route using valid data (b)

This program is a good tool for planning shortest routes. Although program's performance rate is $O(n^3)$, it could be upgraded to as much as $O(n^2)$. Another problem could be that there is no special algorithm in case a given structure of adjacency indicates multi-part graph, so it is possible that program would crash or would not plan the route fully when this kind of data is given.

Path optimization in HIL. Before real test, UAV was tested in Mission Planner auto pilot to find optimal settings for flight by running a hardware-in-the-loop (HIL) simulation over "X-Plane" software. The goal was to find the way-point parameters and auto-pilot controller settings that would give the most accurate and stable flight right over intended path. After a test flight with the default Mission planner settings it was found that they are accurate enough to fly over well-placed way-points set in Mission Planner not deviating too far from electricity lines. In the simulation the plane flew over straight paths perfectly and over way-points placed about 200 meters apart with no sharp turns it flew with minimal error as seen in Fig. 6. On the other hand, the plane deviated a lot from the optimal course on sharp turns taking over 250 meters to get back on



course as seen in Fig. 7.

Fig.6 Straight flight and low-angle turn In these illustrations the red plane icon represents UAV's position at the time the picture was take and the number next to it shows it's altitude. Thin lines coming out from the front of the plane represent some flight data. Red line is current flight direction, orange is pointing at the following checkpoint and black, pink, green are turning guidelines. Green bubbles are way-points and their order is represented by their numbers. Way-points are connected by a straight bright yellow line. And purple line is the simulated flight path. The occasional sharp edges on the flight path are tracking errors due to map shifting and they do not represent actual flight path at these out-of-place points. In these pictures the flight path did not go over some way-points exactly because it was allowed for the plane to miss the way-point by 20 meters. When radius is set too small, UAV has sometimes to turn around due to missing a checkpoint, especially when they are placed too close to one another.



Fig.7 Flight path after a sharp turn.

To eliminate problems after sharp turns, loops around the turns were created [10]. As seen in Fig. 8, placing one way-point 50 meters behind a sharp turn (way-point 5) was not enough to make the aircraft return to the course correctly. Creating a circle of way-points (wp. 10-17) with radius of about 50 meters after a sharp turn turned out to be enough to make the UAV return to the intended path at a correct angle.

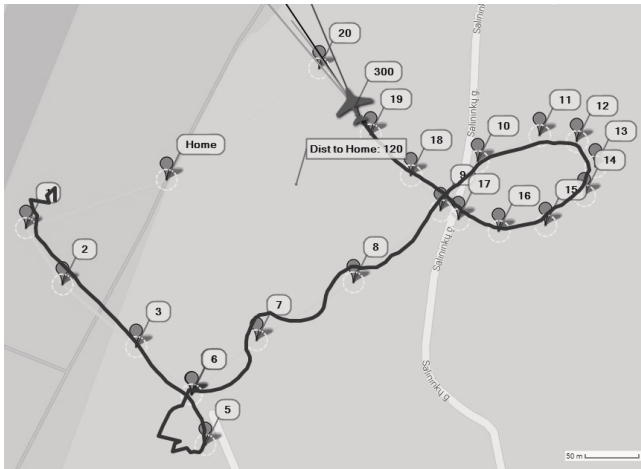


Fig.8 Solution to the Sharp turn problem. According to these results, after smoothing sharp turns with outer-circles, the UAV is able to fly right over the electricity lines with only a small error after turns. Although that will be minimal when inspecting long and straight power lines.

4. Conclusions

It was shown that UAVs could be used for automated power line path tracking with proper software, hardware and take-off / landing area. It would save work time, allows more frequent periodical power line inspections and more responsive critical time inspections. Fully developed system would require minimal human interaction and would eliminate the need for people to travel to hard-to-reach locations just to check for possible faults.

Selected References

- [1] I. Golightly and D. Jones, "Visual control of an unmanned aerial vehicle for power line inspection," 2005, pp. 288–295.
- [2] S. J. Mills, "Visual Guidance for Fixed-Wing Unmanned Aerial Vehicles using Feature Tracking: Application to Power Line Inspection," Queensland University of Technology, 2013.
- [3] J. Zhang, L. Liu, B. Wang, X. Chen, Q. Wang, and T. Zheng, "High Speed Automatic Power Line Detection and Tracking for a UAV-Based Inspection," 2012, pp. 266–269.
- [4] R. Tarjan, "Depth-First Search and Linear Graph Algorithms," *SIAM J. Comput.*, vol. 1, no. 2, pp. 146–160, Jun. 1972.
- [5] U. Brandes, "A faster algorithm for betweenness centrality*," *J. Math. Sociol.*, vol. 25, no. 2, pp. 163–177, Jun. 2001.
- [6] F. Schulz, D. Wagner, and K. Weihe, "Dijkstra's Algorithm On-Line: An Empirical Case Study from Public Railroad Transport," in *Algorithm Engineering*, vol. 1668, J. S. Vitter and C. D. Zaroliagis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 110–123.
- [7] M. Barbehenn, "A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices," *IEEE Trans. Comput.*, vol. 47, no. 2, p. 263, Feb. 1998.
- [8] M. Fleury, "Deux problèmes de Géométrie de situation," 1883.
- [9] H. Fleischner, X.1 Algorithms for Eulerian Trails, Eulerian Graphs and Related Topics: Part 1, Volume 2, *Annals of Discrete Mathematics* 50. Elsevier, 1991.
- [10] B. Carlo L., S. Barbara, and L. Domenico, "Path Planning for Autonomous Vehicles by Trajectory Smoothing using Motion Primitives," *IEEE Trans. Control Syst. Technol.*, vol. 16, no. 6, pp. 1152–1168, Nov. 2008.